Università di Pisa

Scuola di Ingegneria

Laurea Specialistica in Ingegneria dell'Automazione

Tesi di Laurea

Optimal Fusion of 3D Feature Descriptors for Pose Estimation in Robust Grasping Applications

Candidato:

Federico Spinelli

Relatori:

Prof. Marco Gabiccini

Prof. Francesco Marcelloni

Prof.ssa Michela Antonelli

Sessione di Laurea del 03/10/2014
Anno accademico 2013/2014

To my wife, Giulia

Abstract

Pose estimation serves as important tool for robot grasping applications, providing the necessary task-relevant informations about the object that needs to be grasped. Its use provide the robot with an estimation of the object geometry along with a full localization in 6 *Degrees of Freedom* of the object in space. These tools enable the robot to manipulate the surrounding environment and grasp objects within, thus they are the first step towards the realization of autonomous mobile platform.

This thesis makes use of four *global 3D features* to gain multiple descriptions of the same object, then propose a combination of these descriptions, in effort to improve the general performance and robustness of pose estimation procedure. We show how we can acquire meaningful data to build a database of features, so that an indexing and matching procedure can take place, we'll then combine the responses into a list and use it to process our pose estimation.

A closer look on execution time will be kept, so that the pose estimation procedure could be run with *real time* constraints, if need be. The target robot, that will use this procedure, needs to fast identify and localize objects within his environment, in order to competently manipulate them.

Along with data acquisition procedures, we propose some pre-processing pipelines to improve the general quality of our data and we show the benefits of good data pre-processing to mitigate sensor imperfections and noise, that could be affecting the acquisitions.

The pose estimation procedure will be tested in numerous situations, including cluttered environment with both familiar and unfamiliar objects, treating both real and synthetic data, to fully grasp its potentials and also limitations. The thesis is part of the European project Pacman: Probabilistic and Compositional Representations for Object Manipulation, as a mean to establish bases to obtain robust grasping.

Contents

List of Figures

List of Tables

List of Symbols and Notations

This section contains the mathematical symbols and notations frequently used in the thesis.

$\mathbf{p}_i$	a 3D point with $\{x_i, y_i, z_i\}$ as geometrical coordinates
$\vec{\mathbf{n}}_i$	a surface normal at point $\mathbf{p}_i$ with $\left\{n_{x_i}, n_{y_i}, n_{z_i}\right\}$ as direction
$\mathcal{P}_n = \{\mathbf{p}_1, \mathbf{p}_2, \cdots\}$	a set of m nD points or a *Point Cloud*, also represented as $\mathcal{P}$ for simplicity
$\mathcal{P}^k$	the set of points $\mathcal{P}_j$, with $j \leq k$, located in the *k-neighborhood* of a query point $\mathbf{p}_q$
r	the radius of a sphere for determining a point neighborhood $\mathcal{P}^k$
$\bar{\mathbf{p}}$	the centroid of a set of k points defined as $\frac{1}{k} \cdot \sum_{i=1}^{k} \mathbf{p}_i$, also referred as $\bar{\mathbf{c}}$ when it's computed from the whole cloud
$\vec{\mathbf{z}}, \vec{\mathbf{y}}, \vec{\mathbf{x}}$	respectively the $z-axis$, $y-axis$ and $x-axis$ of a reference system
$\vec{\mathbf{v}} \cdot \vec{\mathbf{w}}$	the dot-product between two vectors $\vec{\mathbf{v}}$ and $\vec{\mathbf{w}}$
$\vec{\mathbf{v}} \times \vec{\mathbf{w}}$	the cross-product between two vectors $\vec{\mathbf{v}}$ and $\vec{\mathbf{w}}$
$\|\cdot\|$	the L_2 norm operator, also represented as $\|\cdot\|_2$

g_i^k	the k^{th} element of the i^{th} global feature histogram
$\mathcal{Q}$	the query feature that represent the object pose to be identified
$\mathcal{L}_*$	the list of k candidates to a query for the feature of type $(*)$.
$\mathcal{D}^i$	distance from the query of a candidate at rank i in a list of candidates.

1 Introduction

"I just want the future to happen faster. I can't imagine the future without robots."

Nolan Bushnell (1943)

People in the world are getting older, it is probably expected to double the number of people aged 70 in the next 50 years to come. Since life expectancy is increasing, it is safe to assume, that the chance of people becoming physically and mentally limited, or disabled, is also increasing. This and other problems are what a modern society is facing, thus, as a consequence, science is forced to develop new concepts and technologies for supporting and improving the general well-being of people.

One of the discipline of great interest is Robotics, in constant evolution, this field is in theory capable of developing mobile personal assisting robots to help people in performing their daily activities, such as cleaning the house, set up the table for dinner, cooking, or even load the dishwasher. Robots could improve the general well-being of society in many aspects, such as assist in daily home routines, at work by moving heavy or too big objects, speeding up a production process, operating in hazardous environments or even help explore the bottom of the ocean or the planets in the solar system.

The future were robots takes their part in society is not so distant, as one might think, in fact many recent robotics projects are constantly being developed. To cite a few, the Interaction project [Int14] aims at a continuous daily-life monitoring of the functional activities of stroke survivors in their physical interaction with the environment; the Easel project [Eas13], exploring and developing a theoretical understanding of human-robot symbiotic interaction, and the Pacman project [Pac13], aiming at developing techniques for a robot to grasp and manipulate both familiar and unfamiliar

objects in cluttered environments.

Taking as reference a robot that can grasp and manipulate objects in his environment, the realization of such machine requires it to be equipped with the necessary perceptual capabilities, like vision cameras and haptic perceptions. The robot then has to detect, localize and geometrically reconstruct the object in his environment in order to competently manipulate it. Such tasks, so trivially accomplished by a human, pose a serious problem for the robot.

This thesis is part of the above mentioned Pacman project and aims at overcoming this problem, which is widely known in literature as *Pose Estimation*. With this term it is intended to estimate a position and an orientation (or pose) of an object within the robot environment, this pose may be expressed in terms of a matrix describing a transformation in 6 *Degrees of Freedom* or a translation vector and a quaternion of orientation. In both cases the pose needs to be referred to a reference system known by the robot, for example its base or its head, so that it can efficiently grasp and manipulate the object.

This thesis will cover the problem of estimating an object pose in space and will focus on the visual aspect of the procedure, to do so, it is required to first identify and possibly recognize an object in a given environment, furthermore a semantic of 3D object mapping needs to be adopted to fully describe the object that needs to be grasped. This aspect will be covered in Chapter 2. In Chapter 3, the procedure used to acquire data for the pose estimation is described. We want to create databases of object poses, from which perform indexing and matching, in order to find correspondences with the one we want to estimate. Both synthetic and real data will be acquired, so that we can measure the differences and the impact on the procedure outcome.

Chapter 4 will cover the manipulation of acquired data, finalized to *features* estimation, that will be later used in the pose estimation procedure. This process is mainly meaningful for data acquired by a real sensor, because it is addressed at partially overcoming sensor measurements errors or imperfections, that could lead to a false match of features and consequently an inaccurate pose estimation.

The *features* can be described as some sort of digital signatures that

incorporate the most distinct characteristics of objects they represent. A panoramic of such features is present in Chapter 5, along with tests on their matching performances with synthetic data. These preliminary tests were performed mainly to chose which features we wanted to use, from the current pool. We chose to use four *global* features, among the many, because of their good recognition capabilities and very fast computation time.

To conclude, Chapter 6 will cover the use of such features to achieve a robust pose estimation procedure, used within the Pacman project and will present its performance with a closer look on execution time, since the project mainly aims at grasping objects in real time. The basic idea is to fuse all the feature responses, after matching, to obtain a more robust representation of the unknown pose. This combination can take advantage of each feature strong performance, while suppressing possible mismatch of one feature with backups from the other three.

Finally Figure 1.1 presents and outline of the thesis with optimal reading flowcharts.

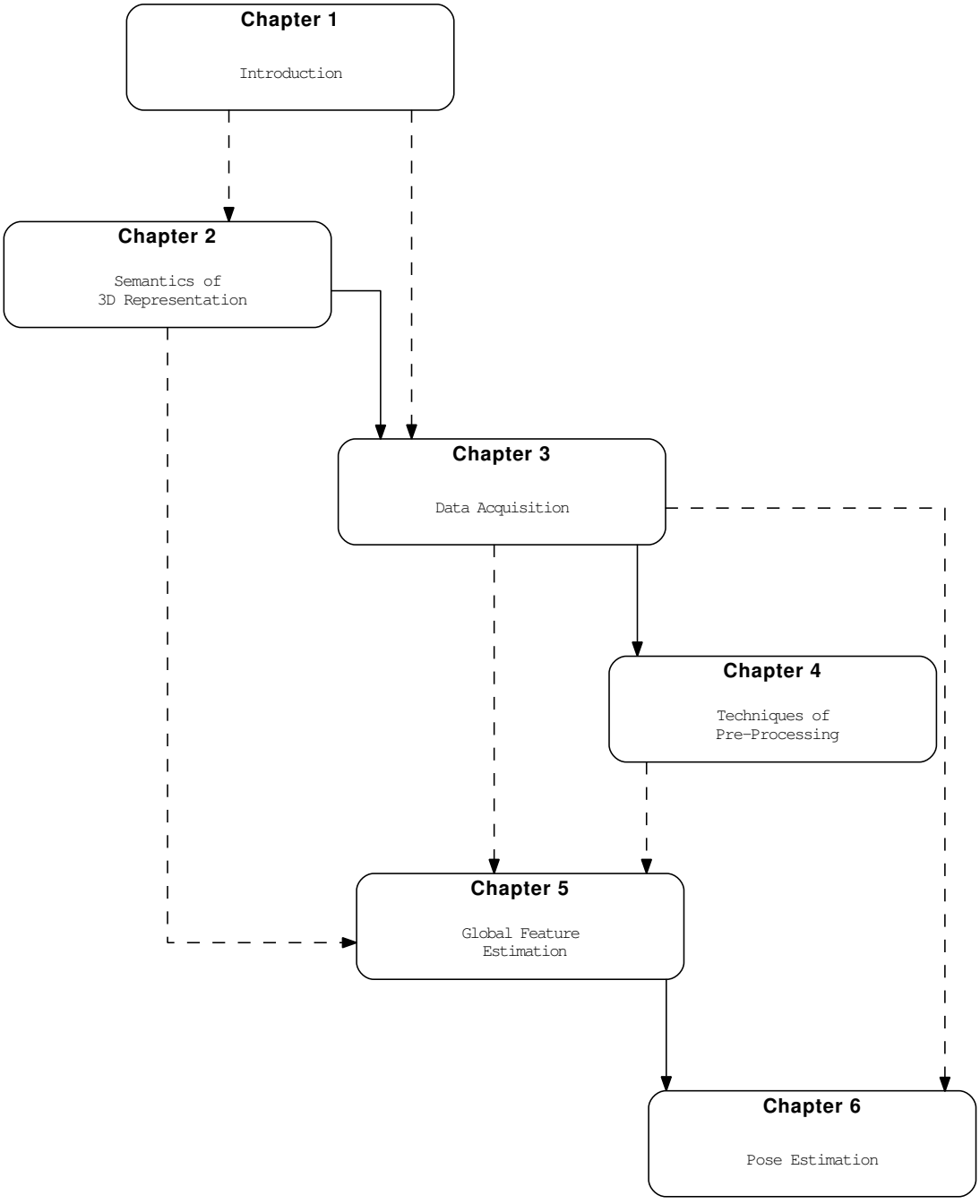


Figure 1.1: Outline of the thesis. The source chapter of an arrow should be read before destination chapter. Dashed arrows represents alternate/optional reading paths.

2 Semantics of 3D Representation

"The more I see, the less I know for sure"

John Lennon (1940-1980)

In this chapter it is presented a semantic to "3D" perception, first introduced in [Rus10, RMBB08b], and adopted also in this thesis. Although the focus of the thesis is not to fully describe this semantic and all its potential, the reader may examine it in depths in [Rus10] and use it in its implementation on [PCL]. Still it's necessary to briefly introduce these semantics in order to understand how "3D" environments can be efficiently described and mapped for the robot to use.

Section 2.1 will talk about why, for applications in need of accurate 3D representations, the use of 2D images should be abandoned, in favor of a more descriptive depth representation.

Section 2.2 will then presents some sensors, used to obtain meaningful 3D representations of the environment and will talk about their advantages and limitations.

Finally Sections 2.3 and 2.4 will pose a few mathematical definitions of what is a 3D representation and how we can define from it, *features* that grasp the fundamental characteristics of objects, or in general the environment.

2.1 The Need For 3D Representations

Humans perceive the environment with images provided by their eyes and so describe the world in terms of what they see. This may erroneously let us think that a robot might be able to address the perception problem with just "2D" images. In reality this could lead to a series of errors, and conduce the robot to fail in capturing the true meaning of the world. The main problems for using flat "2D" images for robot perception are fundamentally the followings:

- Sensitive to noise, such ambient lightening and shadowing.
- Limitations to the data stream; very high resolution streams can occupy too many resources for the robot to process efficiently.
- 2D images are fundamentally a projection of the real 3D world, and may lack the capabilities to describe it fully.

The first two reasons will most likely be addressed with time, as technology progress and new and better hardwares are produced, but the last

Figure 2.1: An example of model matching failure. In the underexposed 2D image none of the features extracted from the model (left) can be successfully matched on the target in the scene (right). The image is taken as it is from [Rus10].

Figure 2.2: An example of a successful match of features extracted from the "beer bottle" template to the scene (left). Unfortunately, by zooming out (right), we see that the bottle matched is in reality just a print on the surface of another different object, in this case a mug. So the procedure has failed to describe the scene geometry. This image is taken as it is from [Rus10].

one is intrinsically unavoidable. To understand better take as examples Figure 2.1 and 2.2, in the first one the lightening in the scene leaves the bottle in front completely underexposed, thus leading to model matching failures. In the second image the model reference gets matched perfectly in the scene, but the "2D" image fails to describe the scene completely, in fact there's no real "beer bottle" in the environment, but just another different object with the model printed on it.

It's clear that the above kind of perception cannot be used in robot grasping applications, and thus it is necessary to adopt a "3D" perception system in order to have an estimate of the image depth and better represent the "3D" world, where the robot operates. In the following section a brief description of the most commonly used sensors to achieve "3D" perception is proposed, in fact it's fundamental to chose the sensor that best adapts to tasks needed to perform.

2.2 3D Sensors

At present day several "3D" sensors with various fields of application already exists, but they can mostly be categorized in two general types:

- Triangulation, or stereoscopic sensors, which estimate distances (or depth) by searching for correspondences between two separated sensors that are analyzing the same scene at the same time. The two sensors needs to know their reciprocal position and orientation so they need to be calibrated with respect to each other.
- Time-of-Flight (TOF) sensors, which emit a signal (light, sound...) towards a surface and wait until it returns to the sensor. The distance between the surface and the sensor is then estimated by measuring the time elapsed, knowing the velocity of the signal emitted.

The first kind of sensors, that try to emulate the functionality of human eyes, are the most difficult to use, due to the need of finding correspondences between data streams. Without going in depth is sufficient to say that is often impossible to estimate good depth measurements for all the pixels in the cameras, so this kind of sensors are probably not the best choice for robot grasping applications.

The second sensor type instead offers enough accuracy for object modeling and relative fast acquisition time, thus rendering them better suited for robot grasping applications. A good variety of these sensors exists, some use lasers as Time-of-Flight signals, like the Swiss Ranger 4000 in Figure 2.3 (right); some others uses infrared or even sound. The sensor used

Figure 2.3: Xtion Pro infrared sensor (left), Swiss Ranger 4000 laser sensor (right), pictures taken from manufacturers website.

Figure 2.4: Some example scenes captured with the Xtion Pro infrared sensor.

Figure 2.5: Acquisitions of sample objects taken with the Xtion Pro infrared sensor, the objects are acquired from different poses with respect to the camera.

in this thesis is the Xtion Pro, visible in Figure 2.3 (left), it uses infrared light and has an integrated rgb camera in order to give color to the acquired "3D" images. An example of some scenes taken with such sensor is visible in Figure 2.4, while in Figure 2.5 some objects acquisitions are visible.

This sensor has a depth accuracy of $3mm$, meaning that points or surfaces distant less than $3mm$ from each other along the depth direction of the camera are not distinguished from one another. This can lead to inaccuracy, in particular when trying to acquire small or thin objects, like for example a fork or a lighter. In addiction infrared light is susceptible to refraction in particular on shining surfaces, like the metal of a pot, or on semi-transparent surfaces like plastics. This refraction noise can sometimes prevent the infrared signal to return to the sensor, leaving the resulting "3D" image with holes or missing parts.

In Chapter 4 we'll discuss how these problems could be partially solved or overcome by applying post-processing to the raw image data obtained from the sensor.

2.3 3D Points And Their Representation

The most intuitive representation for a unit in three-dimensional Euclidean space is a point, $\mathbf{p}_i$. The point has three Euclidean coordinates $\{x_i, y_i, z_i\}$ with respect to a fixed coordinate system, usually having its origin at the sensing device used for acquisition. At the same way we'll call a collection of points, all expressed in the same reference system, as a *Point Cloud*, or $\mathcal{P}_i$ for simplicity. The point cloud structure can then describe a discrete but meaningful representation of "3D" space. The data acquired for "3D" perception is then one or more point clouds that describe the geometrical features of the scene captured by the sensor. Point clouds are widely used in literature for "3D Computer Vision" and the reader is remanded to the Point Cloud Library website[PCL] for more information on the matter.

The Xtion Pro provides several reference systems to chose from and the user can express acquired data in the reference system he likes, but in this thesis we'll use a global fixed reference frame centered in a point of interest of the environment, for example the centre of a table, with $\vec{\mathbf{z}}$ pointing towards the sensor, and with $\vec{\mathbf{y}}$ pointing "upwards". It is important to always have the same reference system throughout all the point clouds acquired, because this way the data is meaningful and coherent with each other; and could be combined in effort to recreate a full "3D" model of the scene or of the object in exam.

A lone point $\mathbf{p}_i \in \mathcal{P}_i$ is insufficient to understand the geometry of a surface, while the whole set $\mathcal{P}_i$ is incapable of distinguish the local features of the surface, such as handles in a mug or a corner of a table. For this reason a subset of points is introduced, by extracting from the whole point cloud some points within a certain distance, or radius r of a query point $\mathbf{p}_q$. This subset of points is called $\mathcal{P}^k$ or *k-neighborhood*, because it includes at most k points in a radius r around the query point $\mathbf{p}_q$. An example of such subset is visible in Figure 2.6. The theoretical formulation for a k-neighborhood $\mathcal{P}^k$ of a given query point $\mathbf{p}_q$ can be given as:

$$\|\mathbf{p}_i - \mathbf{p}_q\|_2 \leq r \tag{2.1}$$

where r is the maximum allowed distance between the neighbor and the query point (the radius), while $\|\cdot\|_2$ is the L_2 Euclidean norm (although

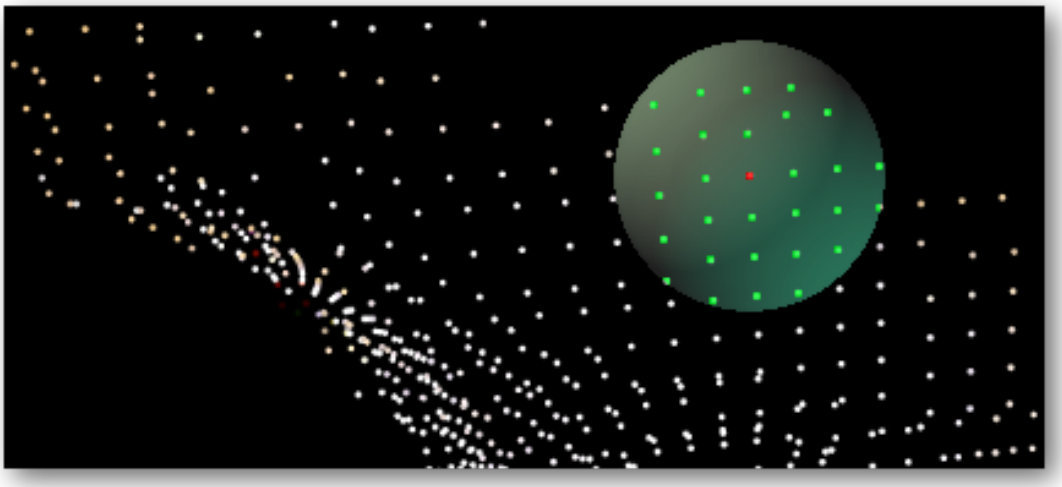


Figure 2.6: A particular of a *Point Cloud* in which is visible the *k-neighborhood* (in green) of a query point (in red).

other different distance metrics can be used). In addiction the number of neighbors in $\mathcal{P}^k$ can be capped to a given value, leaving the subset with at most k points.

The concept of k-neighborhood is important to understand the local geometry of the point cloud and it is used in *Feature estimation* processes, such as *Normal estimation* and many other post-processing techniques that will be discussed in the next chapters.

The key parameter for a "good" subset is the radius r, by choosing a big radius we include more points, thus risking approximating too much the local surface; while with a too small radius there's a risk of having too few points that are incapable of describing the local surface features. In other words it's necessary to find a good compromise between the two extremes. A fundamental help in choosing r can come from the knowledge of the point cloud density. In fact knowing a priori the distance between points can help us calculate approximately how many points there will be in the subset obtained by a certain radius, giving roughly how many "chunks" of points we are extracting, knowing their total number.

In summary a "3D" representation can be expressed in terms of point clouds, or subsets of them, in practice discretizing the environment into points. This collection of points can hold also other informations, such as the color of each point, its normal estimation, or even a user given label, giving space to lots of customizations and practical uses.

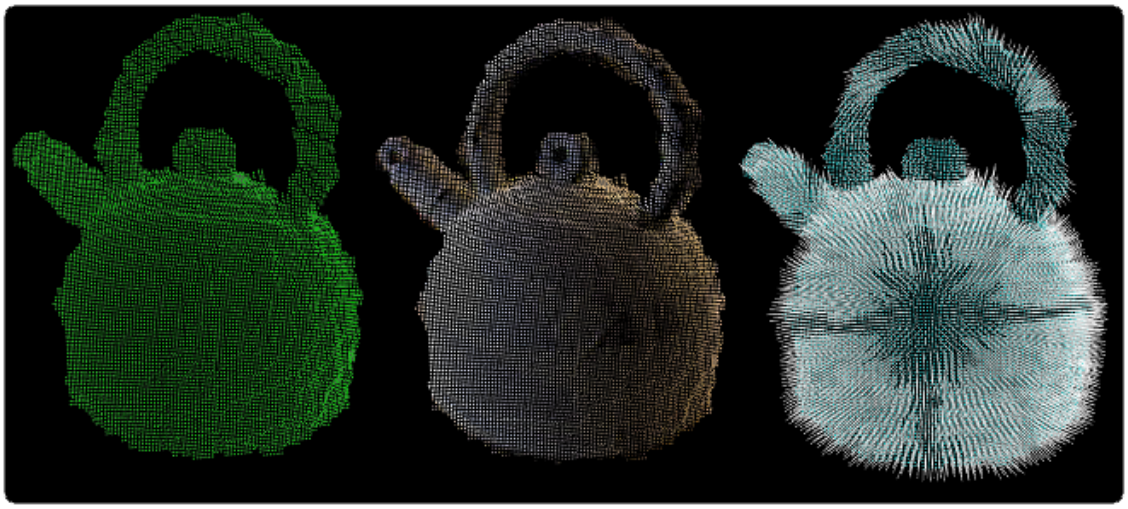


Figure 2.7: Three *Point Clouds* of the same object; the first one (left) contains only the geometrical coordinates of points, the second (center) also includes colors information, and finally the third (right) includes points coordinates and normal estimations.

This notion extends the classical representation of a "3D" point, with only its geometrical coordinates and adds other informations to the point itself, like color, thus making a point cloud $\mathcal{P}$ a generalized container for "nD" points. It is like extending the dimensions of space from 3 to n. Figure 2.7 illustrates three point clouds, all representing the same object, but with different informations stored.

This kind of representation was developed, because many algorithms and procedures often need more informations to compute, other than simple Euclidean coordinates, thus having a compact container to store all these informations simultaneously, is certainly more convenient and memory efficient.

2.4 Characterization of 3D features

The native representation of points, with their Cartesian coordinates, or other informations provided by the sensor such as colors, intensity or even curvature, does not give unambiguous tools to compare or distinguish a point cloud from another.

In fact comparing points and/or their color doesn't reveal informations about the underlying surface in which they lay. For example take two point clouds expressed in the same reference system, one representing a table, the other a coffee mug. Both clouds have many points with the same coordinates and color, but does this say that the mug is somewhat similar to the table?

No it doesn't, the comparison of points as a metrics of point clouds resemblance is an ill posed question. Therefore a different concept needs to take place, over the idea of the singular Cartesian point, that of a *feature descriptor* or just *feature* for simplicity. In literature there's an abundance of different naming schemes, such as *surface descriptor*, *geometric features*, *point shape descriptors*, and so on...

Yet all of them express the same conceptualization, therefore in this thesis they will be referred to as just *features*.

Conceptually, the theoretical formulation of a feature for a given point $\mathbf{p}_q$ and its k-neighborhood $\mathcal{P}^k$, defined in (2.1) can be expressed as a vector function $\mathcal{F}$ in the following way:

$$\mathcal{F}\left(\mathbf{p}_q, \mathcal{P}^k\right) = \left\{f^1, f^2, f^3 \cdots f^n\right\} \tag{2.2}$$

where $f^i, i \in \{1 \cdots n\}$ represents the i-th component of the resultant vector in some space of dimension n, that model the feature of the point $\mathbf{p}_q$, by taking into account also its neighbors in $\mathcal{P}^k$.

With this formulation, the comparison of two points $\mathbf{p}_1$ and $\mathbf{p}_2$, becomes the comparison of the two corresponding features $\mathcal{F}_1$ and $\mathcal{F}_2$ in some n dimensional space. The comparison could be a distance function in some metric, for example L_2 or χ^2. Let then be d the measure of similarity between two features and $\mathcal{D}$ a distance metric, then:

$$d = \mathcal{D}(\mathcal{F}_1, \mathcal{F}_2) \tag{2.3}$$

If d is small, say $d \to 0$, then the two points are considered similar with respect to their features, thus they probably lay on a similar surfaces and

they describe a similar portion of the cloud. On the other hand if d is large, the two points are to be considered distinct, as they probably represent different surface geometries.

The function $\mathcal{F}$ was left undefined, because many features do exists in literature, to cite a few: the Point Feature Histogram (PFH) [RMBB08a], the Signature of Histograms of Orientations (SHOT) [TSDS10], the Spin Images (SIs) [Joh97] and many others. The method of computing the function $\mathcal{F}$ is called *feature estimation* and varies from one method to another, thus leading to different feature space of various dimensionality.

Features presented above are called *local*, because they include the information of the surrounding neighbors around a point of interest $\mathbf{p}_q$ (often called *keypoint*) and thus describe the local information of a portion of the surface of the cloud $\mathcal{P}_i$. Ideally to compare two or more point clouds with *local features*, one has to find a good number of keypoints $\mathbf{p}_q$ in $\mathcal{P}_i$ and for each of them compute the $\mathcal{F}\left(\mathbf{p}_q, \mathcal{P}^k\right)$; then compare the set of features found on a cloud $\mathcal{P}_i$ with the ones found on another $\mathcal{P}_j$.

Based on the number and quality of the correspondences found it is possible to conclude if the point clouds are similar or not, this is not an easy task in itself, but more importantly it could be very computational expensive, leading to high response time. Since for every keypoint it is necessary to calculate the $\mathcal{P}^k$, suppose m keypoints are extracted from a cloud with a method of choice, then the computational complexity of a local feature is at lest $\mathcal{O}(m \cdot k)$, although different features may be faster, or slower.

For this very reason another kind of features were introduced, the *global features*. In the same way as the definition in (2.2) it is possible to define another function $\mathcal{G}$ that estimates a single feature from the whole point cloud $\mathcal{P}$:

$$\mathcal{G}(\mathcal{P}_i) = \left\{g^1, g^2, g^3 \cdots g^n\right\} \tag{2.4}$$

The function $\mathcal{G}(\mathcal{P}_i)$ computes a vector that represent the global feature of the whole cloud $\mathcal{P}_i$, thus easing the process of comparison and matching described above, because a whole point cloud can be mapped into a single feature and comparing clouds results in comparing this feature vector. This is particularly useful in object recognition and pose estimation applications, where the point cloud to analyze is only composed by an object acquisition

or a pose of it.

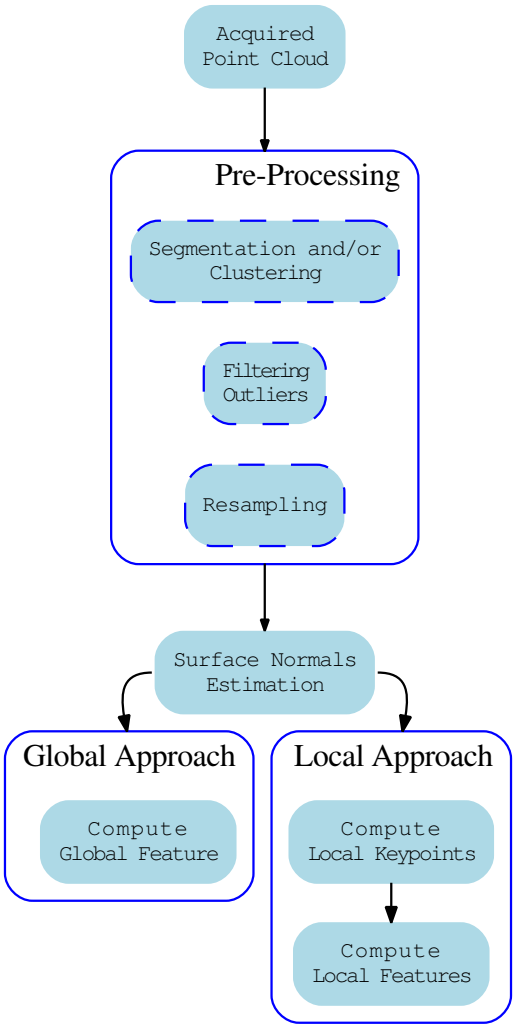
The computational complexity of a global feature is dependant on the total number of points in the cloud, so it is at least $\mathcal{O}(n)$, which is faster than their local counter part, but the major computational time is gained during the matching phase. This is because only one feature is produced per cloud, instead of m, thus the matching process is simplified and faster.

Generally speaking, for the process of features estimation is often necessary to compute several pre-steps, some of which mandatory. They may include a pre-processing to treat acquired data to reduce noise or modify the cloud in some manner, however most of the features requires a pre estimation of *surface normals* that will be described, along with other pre-processing steps, in Chapter 4.

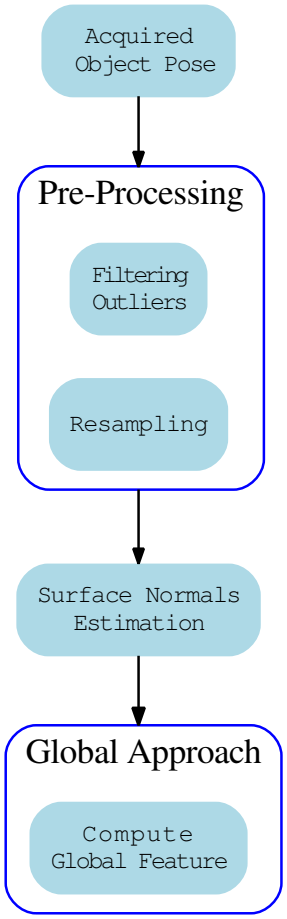
This leads to the computation pipeline visible in Figure 2.8a. Although the illustrated process represent a common pipeline for most applications, in this thesis we chose a slightly different one, visible in Figure 2.8b.

The proposed pipeline is composed by the all optional pre-processing steps and follows a global approach for features estimation, meaning that we chose the implementations of the global features defined in (2.4). The main reason for this choice is the need of a reasonably fast application to efficiently compute the pose of objects in order to grasp them with the robot, so we decided for the "faster" global approach in order save precious computation time, at the expense of losing the local, and probably more accurate, description.

In Chapter 3 we will discuss how we can acquire point clouds to generate a database for pose estimation, thus achieving the first step of the pipeline.



(a) General pipeline.



(b) Pipeline adopted in the thesis.

Figure 2.8: The general features' estimation pipeline (a), and the one adopted in this thesis (b). Dashed nodes represents optional pre-processing steps.

Data Acquisition

3

"It is a capital mistake to theorize before one has data"

Arthur Conan Doyle (1859 - 1930)

Data acquisition plays a fundamental role in recognition and pose estimation applications and it is necessary to acquire coherent and meaningful data in order to expect good results. The focus of the thesis is to achieve a pose estimation of different objects for grasping, and thus the point clouds were acquired with this goal in mind. Particularly we are interested in objects viewed from different viewpoints with respect to the sensor, in order to simulate a robot that is viewing a scene from various viewpoints.

The robot needs to understand if what is seeing is a pot or a mug and then estimate where and how the object is posed in its reference system, so it can grasp it. To achieve this, various databases of objects were built in order to have correspondences with the "real" environment.

This chapter describes how these databases for pose estimation were obtained. Fundamentally, two kinds of data were created using two different techniques, that can be summarized as follows:

1. Using a virtual sensor to acquire point clouds from object models, created from polygonal meshing techniques and modelled exactly as the *real* objects. Various point clouds were obtained from a variable virtual viewpoint around the object, expressed in terms of latitude and longitude coordinates. This kind of database is called *Synthetic Database* and its creation is covered in Section 3.1.

2. Using the real Xtion Pro sensor in conjunction with a rotating table, to acquire various "real" object clouds at precise angular displacements

of ten degrees. This database is called *Real Database* and its creation is covered in Section 3.2.

3.1 The Synthetic Database

The creation of the synthetic database makes use of a new software, built specifically for it. The synthetic scanner can acquire multiple point clouds from a polygonal mesh from different viewpoints around the object in latitude, longitude coordinates. Although scanner softwares like this already exist, they didn't quite fit our needs, like the necessity to acquire clouds from precise viewpoints, or the ability to set the resulting cloud resolution; so a new one was developed entirely from scratch.

The "new" software works with polygonal meshes, so a series of object models were also created to work with this scanner, some of them are visible in Figure 3.1. In order to have consistency between the various point clouds acquired, every mesh has its own reference system centered on an imaginary table at the bottom of the objects.

The viewpoint of acquisition around the mesh can be specified by giving a radius, a latitude and longitude angle to simulate all the possible positions the sensor could be placed in a sphere around the object. This approach can give us clean point clouds taken from various viewpoints that can be used to build our synthetic database.

To accomplish this task the program loads a mesh model into memory and creates a rectangular grid of points, perpendicular to the viewpoint direction and behind the object so that the objects stays in the middle between the grid and the viewpoint; then it projects virtual lines from the

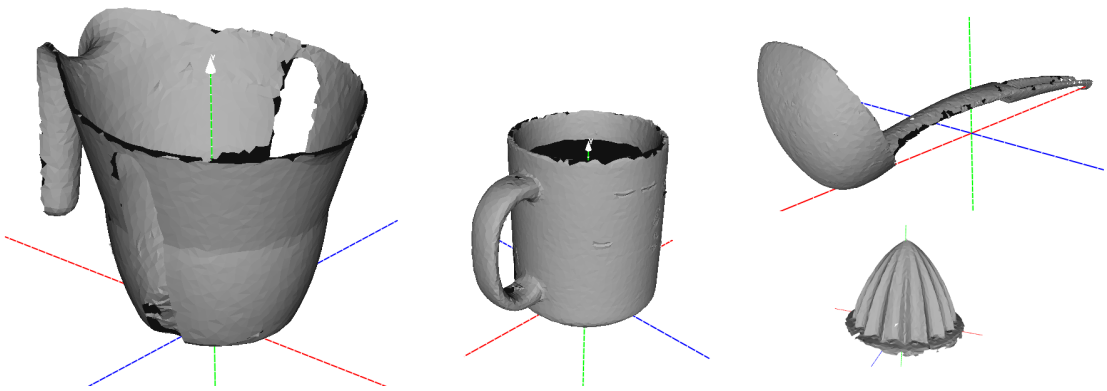


Figure 3.1: Polygonal meshes of some objects with their own reference system.

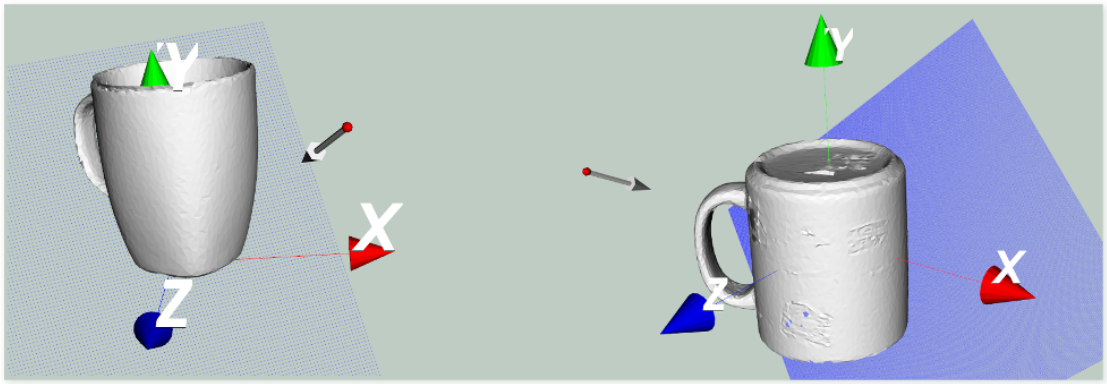


Figure 3.2: Two example acquisitions, made with the synthetic scanner software, note the red dot with the arrow that represents the viewpoint of acquisition, while the blue dots behind the object represent the grid.

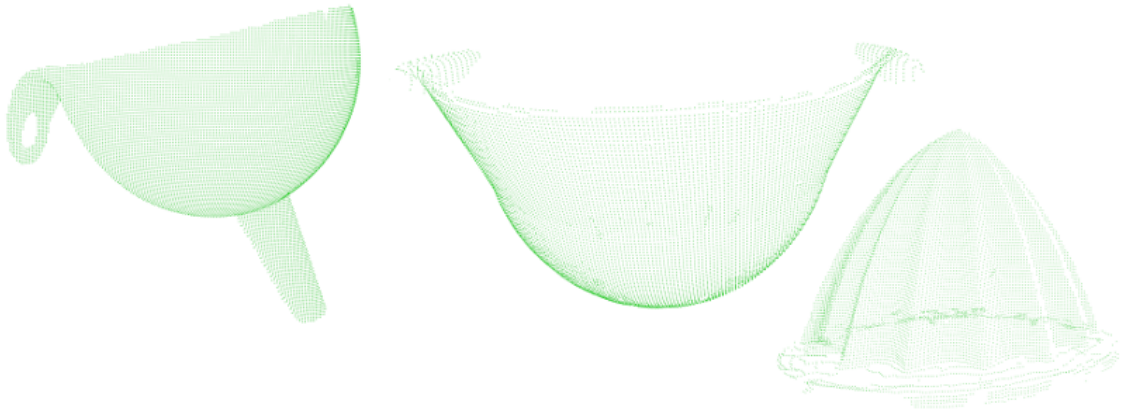


Figure 3.3: Examples of acquired point clouds with the synthetic scanner.

camera to every point in the grid and finds intersections between these lines and the object mesh, every first point found is then saved into the resulting point cloud.

The software also posses an interactive visualization to view the parameters set, done in VTK (The Visualization ToolKit) [VTK], to ease the user setting the various parameters that control the output point clouds. An example of the program in action is shown in Figure 3.2, while some output point clouds are visible in Figure 3.3.

The grid of points is the core of this piece of software, in fact its dimensions and resolution control the quality and density of the output point cloud. In our acquisitions we made sure that the grid was big enough to catch the object dimensions, but also dense enough to make the resulting point cloud with quite a lot of points. We wanted very detailed clouds to best catch the objects features, because these acquisitions still have to

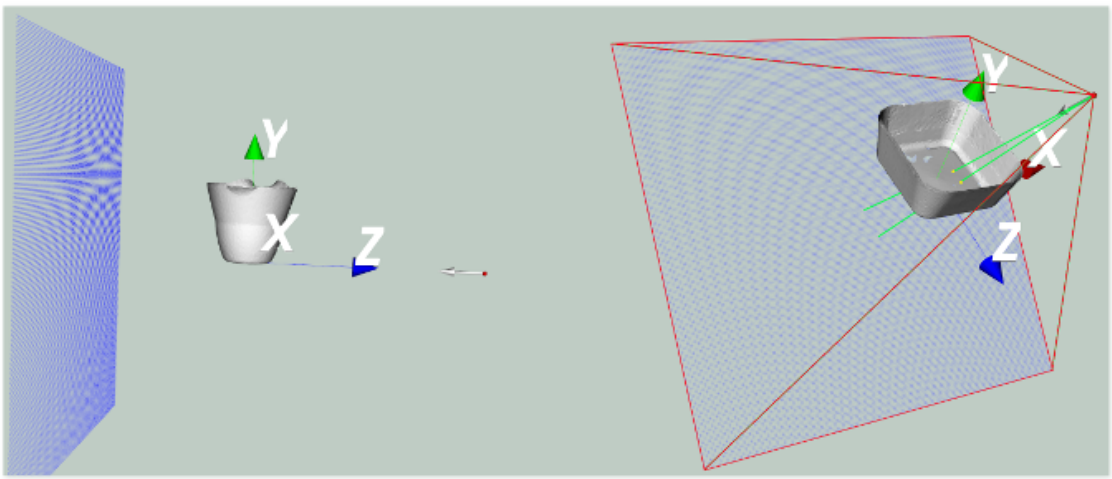


Figure 3.4: Grid placement in the synthetic scanner software (left), the grid is normal to the viewpoint direction and placed behind the object at user defined distance. The other picture (right) shows the boundary pyramid of the virtual sensor in red, while in green are shown some intersection lines between the sensor and the grid, resulting in the correspondent points in the output cloud (yellow dots).

be processed with filtering and sampling techniques that reduce the total number of points.

In Figure 3.4 one can see the grid placement and the process of intersection between the virtual lines and the object, that creates the output cloud. It is important that the objects fit inside the projections of the viewpoint on the grid borders, represented by the red pyramid in figure, otherwise parts of the object may fall outside the virtual camera range and thus do not appear in the output cloud. To this end the software is capable of move away, resize and roll the grid to the user needs.

Since the main goal of building the synthetic database, apart from helping us chose the best *features functions* to apply on real data, is to find correspondences in features from the synthetic data and real one, the synthetic data needs to approximate reality with decent approximation. If the above was true, lots of acquisitions could be made offline with the use of this software, saving lots of effort in building a large dataset of poses.

The data generated with the virtual sensor, however, is too perfect, reality is affected by noise and depth measurements errors, resulting in too different clouds. In effort to ease this problem, but not to resolve it completely, a new method of acquisition was developed and added to the one described so far.

The main idea behind this method is that the real acquisitions appears

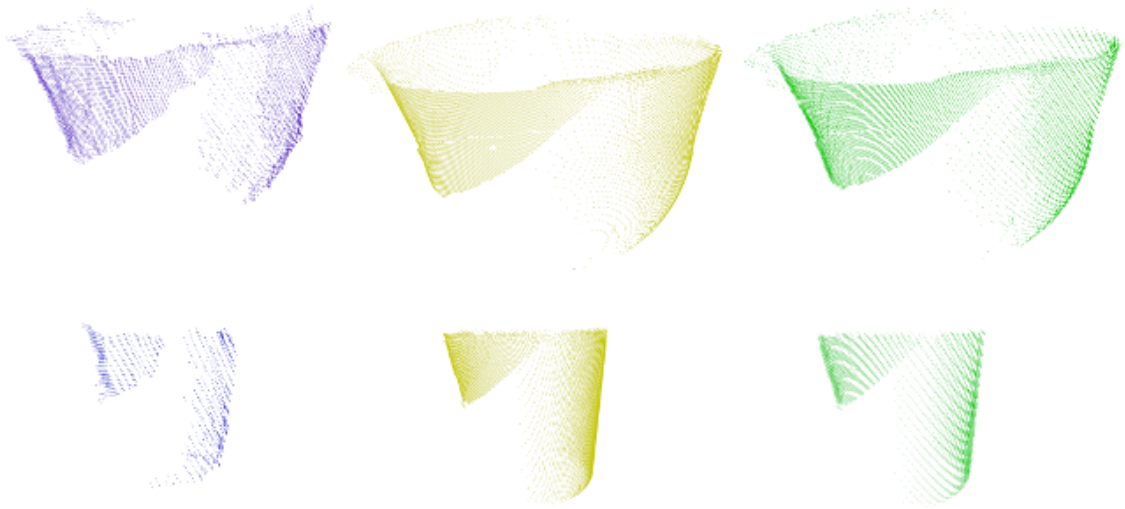


Figure 3.5: Acquisitions of point clouds: with a real sensor in blue (left), with the virtual sensor in yellow (centre) and with the virtual sensor implementing the "plane slicing" technique in green (right).

as "sliced" by planes perpendicular to the sensor viewpoint, this is caused by the Xtion Pro depth resolution of $3mm$. In reference to Figure 3.5(left) one can see that the objects, representing a container and a glass, look like "sliced" by planes and they are not perfectly smooth along their depth direction.

So we decided to apply this "plane slicing" also to the synthetic data generated from the virtual sensor and we acted in the following way:

- Upon obtaining the output point cloud, a further process takes place, consisting of subdividing the cloud into smaller areas contained between two consequent parallel planes both normal to the viewpoint direction.
- All points found in those areas are then projected on one of such planes according to the distance from it.
- This process is then repeated all along the viewpoint direction until no more "unprojected" points can be found.

Figure 3.6 illustrates this process. Adjusting the distance between consecutive planes let us set a *depth noise* similar to the one of the real sensor has. The Figure 3.5(right) shows point clouds obtained with the virtual scanner applying this method, one can see the difference from the ones that doesn't implement this method (centre).

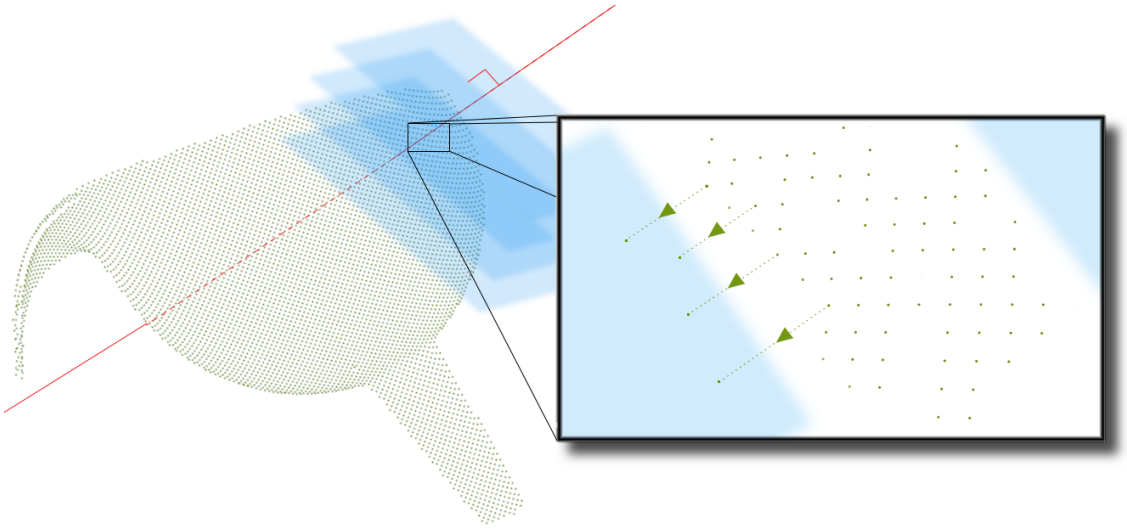


Figure 3.6: Illustration of *plane slicing* technique; the point cloud is sliced with consecutive planes (azure), all normal to the viewpoint direction (red line). All points found between two planes are projected on the nearest of them (green dotted arrows).

Finally to obtain the synthetic database we acquired all the object models we had, applying the "plane slicing", with a latitude fixed at 30 degrees, a distance from the virtual camera set to 80cm and a variable longitude spawning 360 degrees with pass of 10 degrees for each scan, thus totalling 36 scans per object.

We chose this approach to have a database containing comparable scans to the ones of the real database, describe below. So that we could operate matches between the two, if needed.

3.2 The Real Database

In order to have more and different data to perform tests and to have a database that better respect reality, we also acquired data with the Xtion Pro and a rotating table.

The rotating table for acquiring object poses is composed by a servo-motor, an encoder and an electronic board for control. The motor can rotate a round wooden plane fixed to it by 360 degrees in both directions, while its position is tracked by the encoder that can measure the angular displacement. A Picture of the table and the Xtion mounted on the robot can be seen in Figure 3.7.

Figure 3.7: The rotating table (top-left), the Xtion Pro mounted on the robot, pointing at the object (right), and the resulting acquired point cloud (bottom-left).

The software developed can control the table to achieve incremental angular displacements with great precision and speed; for the purpose of acquiring multiple point clouds of objects from various point of views with the Xtion Pro, the table was fully rotated by 360 degrees from its initial position with steps of 10 degrees each. The sensor was mounted at distance of circa 80 cm from the table centre and positioned to have a latitude displacement of 30 degrees from the table plane. This way for every step a new point cloud is acquired, thus totalling 36 acquisitions per

object, each representing a different view. This poses are then processed to remove the table from the scene, to have point clouds representing only the objects.

It is also necessary to have a precise model of the table on top of which the object is placed in order to remove it and also to refer all the clouds to the same reference system to achieve consistency between poses.

The idea is then to obtain a reference frame of the table respect to the sensor and then express each point clouds acquired to that reference frame. This method will ease our recognition and pose estimation process and will also produce a database consistent with the synthetic one, since also that was similarly obtained, see Section 3.1. Summarizing, the main steps to acquire poses of the objects can be expressed as follows:

1. Acquire a model of the table and compute the transformation to the sensor reference frame to this new frame centered in the table.
2. Start acquiring the object poses and apply to all of them the transformation found on *step 1*.
3. Remove the table from each scene.

The following sections explain point by point this process of acquisition for the real database.

3.2.1 Table Model

The procedure to acquire a model of the rotating table manly consists of finding a circle in "3D" space, since the table is round, and construct a reference frame on its centre. The algorithm implemented, to solve this problem can be summarized by the following steps:

1. Acquire a point cloud of the table without any object on it.
2. Crop the resulting cloud around the table, in order to remove the background.
3. Find a circle in 3D space with the RANdom SAmple Consensus (RANSAC) algorithm.
4. Estimate the circle centre and the normal of the plane on which it lays.

5. Build a reference system in the centre of the table, with $\vec{\mathbf{y}}$ corresponding to the plane normal found on the previous step.

6. Calculate a transformation matrix to express the sensor reference frame to this new one.

7. Save the transformation to disk for further use.

The point cloud acquired during *step 1* is visualized by the program and the user is asked to find the table in the scene, by clicking on its centre. This non-automatic procedure is mainly done to speed up the algorithm and ease *step 2*. The point cloud is then cropped with a pass through filter, previously modelled to just include the table and nothing else, this is necessary to prepare for *step 3* and guarantee its success.

If the steps so far are successful, the acquired cloud only contains the table devoid of the background. In *step 3* the RANdom SAmple Consensus (RANSAC) algorithm, introduced in [FB81], is used to find a circle in "3D" space, this is guaranteed to converge at the table borders, since in the cropped point cloud the only circle present is the table itself.

Once the circle is found, the coordinates of its centre, the radius and the normal to the plane in which it lays are returned by the algorithm itself (*step 4*). With these informations a reference frame is built in the table centre, with $\vec{\mathbf{y}}$ corresponding to the table normal, $\vec{\mathbf{z}}$ pointing towards the sensor and $\vec{\mathbf{x}}$ as the cross-product of the other axis.

In *step 6* the transformation matrix between the reference frame of the sensor and the table is computed by concatenating a translation and a rotation. The first is simply obtained by translating to the table centre computed in *step 4*, the latter is found with Axis-Angle notation. Let $\vec{\mathbf{y}_0}$ be the $y-axis$ of the sensor reference system and $\vec{\mathbf{y}_1}$ be the one of the table, then the angle θ between these two vectors can be computed with:

$$\theta = arccos\left(\frac{\vec{\mathbf{y}_0} \cdot \vec{\mathbf{y}_1}}{\|\mathbf{y}_0\| \cdot \|\mathbf{y}_1\|}\right) \tag{3.1}$$

where $\|\cdot\|$ is the magnitude of the vector. The axis of rotation between the two vectors, let it be $\vec{\mathbf{a}}$, can then be calculated with the cross-product:

$$\vec{\mathbf{a}} = \vec{\mathbf{y}_0} \times \vec{\mathbf{y}_1} \tag{3.2}$$

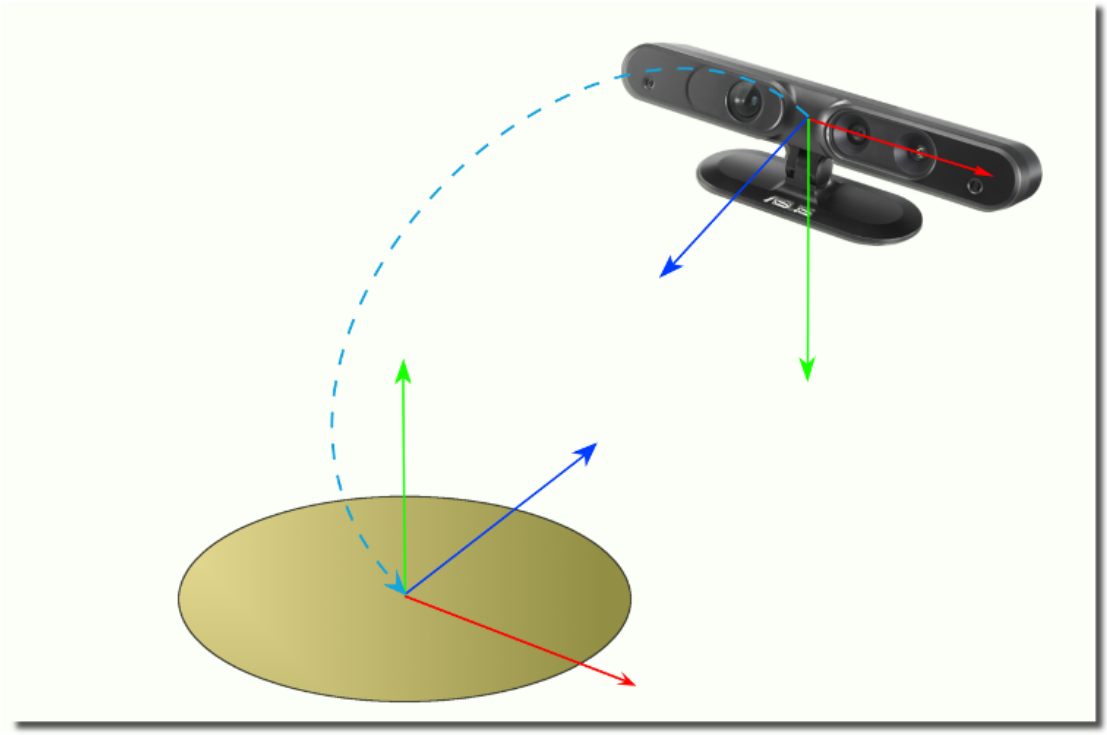


Figure 3.8: The two reference frames: the one of the sensor and the newly created one on the table centre. The transformation between the two frames is represented by the azure dotted line.

The rotation can be expressed in terms of Axis-Angle representation and then written as a matrix $\mathbf{R} \in SO(3)$ with the Rodriguez formula:

$$\mathbf{R} = \mathbf{I} + sin(\theta)\mathbf{A} + (1 - cos(\theta))\mathbf{A}^2 \tag{3.3}$$

where $\mathbf{I}$ denotes the identity matrix of dimension 3 and $\mathbf{A}$ is the *cross product matrix* of $\vec{\mathbf{a}}$.

Final roto-translation matrix can finally be computed in *step 7*, by concatenating the rotation and translation found, and then saved onto disk, along with the radius of the table to be further used in the acquisition process. The two reference frame and the transformation are visible in Figure 3.8.

However with this method it is not guaranteed that $\vec{\mathbf{y}}_1$ will actually point "up" the table, in fact only the direction of the axis will be correct, but the orientation may not. To work around this issue a further control is added to the procedure between *step 4* and *5*, which consists of checking the table normal orientation and flip it if necessary.

3.2.2 Objects Acquisitions

With the table model and transformation saved, it's now possible to begin acquiring the actual object point clouds, in this section this process is described.

First the rotating table is programmed to make a 360 degrees rotation with 35 stops of 10 degrees each on which the Xtion sensor was made to acquire the poses of the object placed roughly on the centre of the table. The rotating table was made perform each step in about 5 seconds in order to minimize the vibrations and sudden movements so that the object is not moved accidentally during the acquisition. To further enhance this procedure and to minimize human effort so that the process of object acquisition was the most automatic possible, an autonomous procedure to separate the object scan from the surrounding scene was developed. The procedure of object acquisition can then be summarized with these steps:

1. Acquire a point cloud of the object and the surrounding scene from the current view.
2. Transform the point cloud with the table transformation saved during the table acquisition steps.
3. Crop the cloud so that only the object and the table remains.
4. Find the points that lay on the table with the RANSAC algorithm and eliminate them.
5. Further filter the cloud to remove spurious outliers that may be left after the table removal.
6. Transform the cloud by rotating back around $\vec{\mathbf{y}}$ by the angle the table has covered so far.
7. Rotate the table by 10 degrees, and repeat from *step 1* until all 36 poses are acquired.
8. Revise and adjust all the acquired scans, to check for errors, and repeat some of them if necessary.
9. Rotate back the table to the zero position and change the object to be acquired, repeat all steps.

The acquired point cloud catches a particular view of the object along with all its surroundings like walls, floor and other objects and tables so it's important to remove from the cloud all the uninteresting points. To do this automatically the saved table transformation is used, with that it is known the centre of the table and by consequence the object position in the cloud.

After transforming the point cloud reference frame with the table transform (*step 2*), the cloud centre is now roughly in the middle of the table and $\vec{\mathbf{y}}$ is normal to it pointing towards the ceiling. Since also the table model radius was saved, it is possible to crop every point that is outside a certain distance from the table centre, experimental results showed that a distance of two times the table radius was enough to catch safely all objects without risking to accidentally crop them.

Then after *step 3* in the cloud remains only the object scan and the rotating table underneath it, all other surroundings in the scene were removed by the cropping. At *step 4* a safe assumption is made: the biggest planar surface, whose normal is very similar to $\vec{\mathbf{y}}$ of the new reference frame, is only the table plane. This assumption is true for most objects placed on the rotating table, but not for all. For example a big planar tray or a big dish may blend with the table surface and thus they can not be separated from the table automatically. For objects like those a manual inspection and segmentation was necessary and so they were acquired with the table underneath them and separated from it a posteriori. Instead, for all other "standard" objects like cups, containers, funnels and various utensils, the assumption made holds, so *step 4* takes place.

A RANSAC algorithm starts searching for all points on a plane, whose normal is $\vec{\mathbf{y}}$ with a certain tolerance, hence it searches for the plane of the rotating table. The algorithm converges at the biggest plane with those characteristics and thus if the assumption made before is true it converges on the table plane, finding all the points that lay on it, leaving the object unscratched. Once identified, the points are removed from the cloud leaving only the object. In Figure 3.9 are visible some object acquisitions prior the table removal and after it.

Experimentally, sometimes a few inliers on the plane are not classified as such by the RANSAC algorithm and thus not deleted, this may occur because some points may end too distant from the plane table model due to

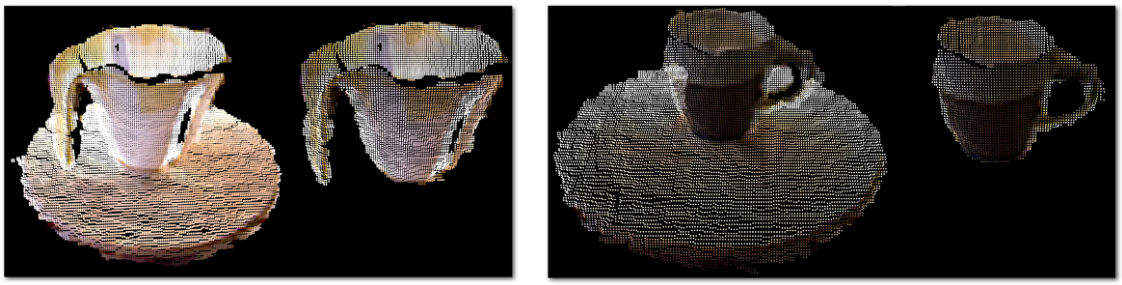


Figure 3.9: Some object acquisition process: a container (left) and a mug (right). In both images are visible the object prior the table removal procedure (left part) and after it (right part).

noise on the sensor or lightening interference. To account this unpredictable issue *step 5* is performed, it consists in filtering the point cloud with a statistical outlier removal filter. This filter calculates the mean distances of points and computes a Gaussian distribution over it, all the points that are farther from the mean distance plus the standard deviation are considered outliers and thus removed. This filter is part of the *pre-processing* steps and will be further described in Section 4.1 in the next chapter. The filtering process efficiently removes all those spurious points left from the RANSAC algorithm while leaving the object intact, because it's composed of dense points.

Finally in *step 6* the object acquisition is rotated backwards around $\vec{\mathbf{y}}$ by the same angle the table has rotated so far, meaning that, for instance, if the rotating table is currently at an angular displacement of +30 degrees the object is rotated by -30 degrees.

This step is done to try to align all the scans together so that if one imagines plotting them all in the same picture he would see them overlap and form a complete object model. This process is useful to achieve consistency between all clouds of the real and synthetic database, and effectively we obtain clouds that look like the sensor was moving around them while the object was still, exactly like the process of acquisition in the synthetic database, Section 3.1.

As a final control, a software for re-visioning all the point clouds of the objects was developed and it's used to achieve *step 8*. This software cycles through all the scans and lets the user manually crop the cloud as needed to account for some imperfections that may have arisen during the previous steps of the procedure. Normally most of the clouds visioned had

Figure 3.10: The objects that compose the *real database*.

nothing to revise and thus were just saved to compose the real database, while others had to be refined.

As a further note, some objects needed to be covered with opaque films to reduce reflections of shining and semi-transparent surfaces, that interferes with the sensor infrared rays, leading to unusable noisy point clouds.

The *real database* is then composed of 36 acquisitions of 10 degrees pass for each object, some of which are visible in Figure 3.10, and it is thus composed of circa 800 point clouds.

This acquisition process was repeated three times in different periods of time, in order to cover most of the possible lightening conditions in the room, that may alter the sensor performances. The three real databases thus represent the same objects in the same poses, but the clouds composing them are slightly different, due to illumination and also inevitable human imprecisions in positioning the object on the rotating table. These inequalities render the databases perfect candidates for numerous tests, whose results are exposed in Chapter 5 and 6. To clarify we wanted to find correspondences in the features space between an object in a database and the same object in another one.

Techniques of Pre-Processing 4

"Nothing is particularly hard if you divide it into small jobs"

Henry Ford (1863 - 1947)

In this chapter it is described the procedures of *pre-processing*, applied to the data before the actual computation of *features*. In reference to Figure 2.8b on page 16, the procedures of outliers filtering and resampling will be discussed in Sections 4.1 and 4.2.

Even if a procedure of segmentation is used during the database acquisitions, see Section 3.2.2, it's not the purpose of this thesis to fully analyze all the various aspects of this field of study, which is very vast and under constant evolution. For these reasons, what we exposed in the above mentioned section is enough explanation on the matter, the reader may refer to [Rus10] for further clarifications.

The need for our application to handle point clouds, that represent only the object we want to analyze and nothing else, is clarified by the choice of implementing *global features*. We want a feature to globally describe the object acquisition and thus we can't have any background that would interfere with such representation, inevitably leading to apply techniques of region segmentation and clustering. This part of the procedural pipeline is then included and performed directly during the data acquisition phase.

Instead the process of *normals estimation* will be described in this chapter, and more specifically on Section 4.3, because it's a common step for almost every features representations, be they local or global and it's of fundamental importance for understanding how the global features are estimated from such normals.

4.1 Outliers Filtering

The procedure of outliers filtering removes points, previously classified as "outliers", from a point cloud. An outlier is precisely a point that is distant from other points and does not fit on the currently outlined surface, these points can disturb the estimation of features and normals, because they may appear in the *k-neighborhood* of a certain query point.

Outliers can arise due to sensor noise or simply bad lightening conditions, even if it's not strictly mandatory to remove them from the data, it's still a good idea to do so, because it generally improves the quality of the point cloud and consequently its feature estimation.

Although many methods for removing outliers exists, the one chosen in this thesis perform a statistical analysis of the data, and then classifies points as inliers or outliers. For this reason, it is called *Statistical Outliers Filter* and its main steps can be described as follows:

1. Calculate a k-neighborhood, with a chosen k, for every point in the cloud.
2. Perform a statistical analysis, by calculating *mean* and *standard deviation* of point distances in the k-neighborhood.
3. Classify points as inliers or outliers, based on their mean distance inside their k-neighborhood.
4. Removes the points classified as outliers from the point cloud.

The k-neighborhoods, calculated in *step 1* are obtained by including the k nearest points around the point of interest, or query point, and then repeated for every point in the cloud. In *step 2*, at every point gets associated a mean distance, that is calculated as the mean of the distances between the point itself and every other point found inside its k-neighborhood. Let this mean distance associated to the point $\mathbf{p}_i$ be $\bar{d}_i$. A statistical distribution of these mean distances is calculated and from it a mean (μ) and standard deviation (σ) are computed. The algorithm then checks if $\bar{d}_i$ is included in the interval $\mathcal{I}$ defined as:

$$\mathcal{I} = [\mu - \alpha \cdot \sigma, \quad \mu + \alpha \cdot \sigma] \tag{4.1}$$

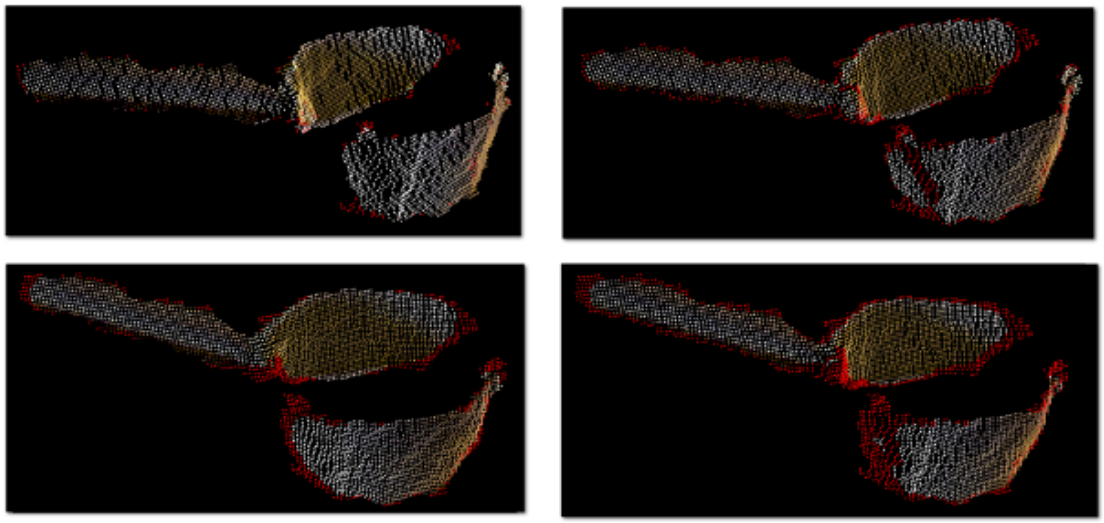


Figure 4.1: Outliers filter applied to a point cloud of a pan, with different parameters: $k = 50$ $\alpha = 2$ (top-left), $k = 50$ $\alpha = 1$ (top-right), $k = 150$ $\alpha = 1$ (bottom-left) and $k = 150$ $\alpha = 0.5$ (bottom-right). The points colored in red are the ones classified as *outliers*.

where α is a weight that multiplies the standard deviation to control the width of the interval.

A point $\mathbf{p}_i$ gets classified ($step3$) according to the following rule:

$$\mathbf{p}_i \in \begin{cases} \{inliers\} & \text{if } \bar{d}_i \in \mathcal{I} \\ \{outliers\} & \text{if } \bar{d}_i \notin \mathcal{I} \end{cases} \tag{4.2}$$

Finally all points classified in $\{outliers\}$ get removed from the point cloud ($step4$).

By controlling the k and α parameters it's possible to make the filtering process less or more restrictive, in particular by modifying k one can vary the $\mathcal{P}^k$ of $\mathbf{p}_i$, thus altering the mean distances $\bar{d}_i$ and consequently μ. While, by modifying α one can alter the interval $\mathcal{I}$, rendering it less or more wide. An example of this parameters tuning is visible in Figure 4.1, where filters with different parameters gets applied to a point cloud.

As a final note, an example of the Statistical Outlier Filter capabilities is shown in Figure 4.2, where a kitchen scene gets filtered to improve its general quality by removing spurious outliers visible in the circled zones. The parameters in this filter are set to $k = 30$ and $\alpha = 1$.

Since our data represents mostly small objects and they are not so affected by outliers, we decided to apply a mild filter with $k = 30$ and $\alpha = 3$, to remove spurious outliers mainly derived by depth measurement errors

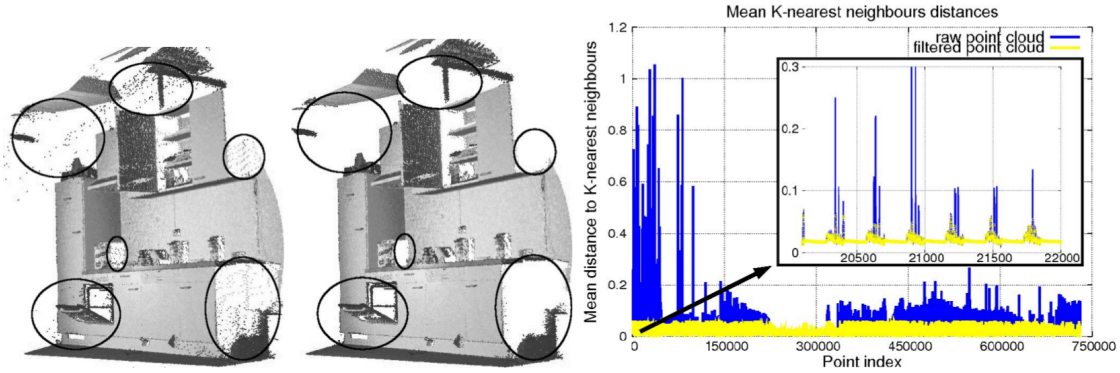


Figure 4.2: Example of the statistical outliers filter procedure: a point cloud with many outliers (left), the resulting point cloud after the filter (center) and a graph with mean point distances (right). Image is taken from [Rus10].

around the object borders. This kind of process is pictured in Figure 4.3 applied to a point cloud of a plug. As one can see, only a few outliers get removed, they are represented as the red points in the picture. This filter then mainly refines borders leaving the bulk of the object intact, since it is composed of dense points.

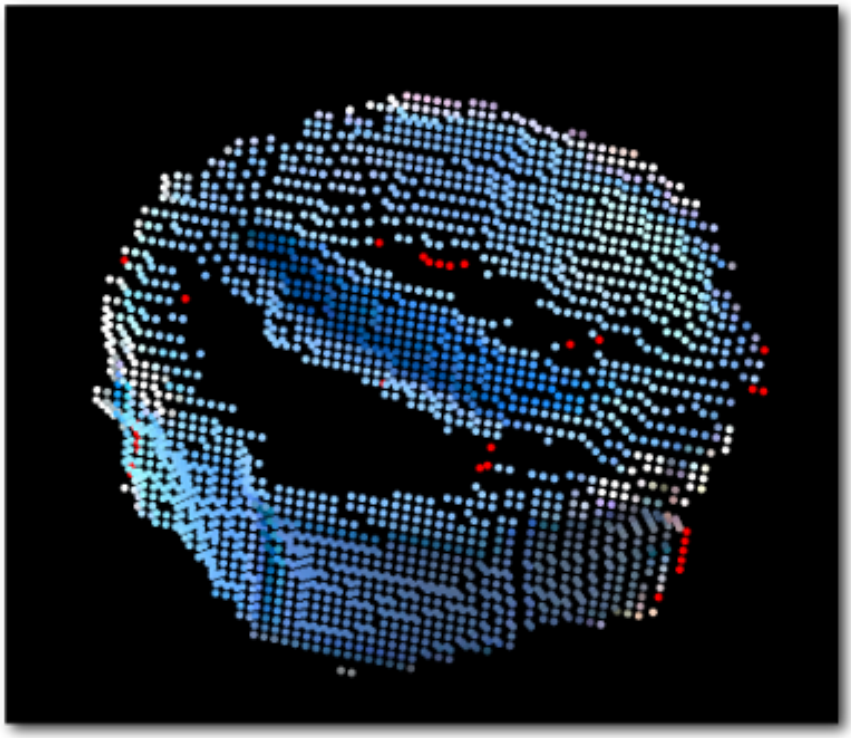


Figure 4.3: Filtering procedure applied to our data. Only a few outliers (red points) gets removed from the plug point cloud, leaving most of the object intact. This filter can then be considered to act as a border refinment that tries to smooth them.

4.2 Data Resampling

A resampling is a method that changes the density of a point cloud, by adding, removing, or simply moving points and equally distribute them along the surface, generally speaking if the resulting point cloud has fewer points than before, then we have downsampled it, otherwise we have performed an upsampling.

In "3D" feature estimation it's generally more useful a downsampling rather than an upsampling, because the acquired point cloud has already a sufficient number of points to estimate the features, so reducing its number can speed up the computations, while maintaining a uniform density of them.

Having a uniform, and more importantly known, density of points can be beneficial to the features's estimation processes, that make use of k-neighborhood subsets. In fact, by recalling (2.1), a fundamental help in choosing the radius r to define $\mathcal{P}^k$ can come by knowing the average density of points in the clouds, because setting a radius will automatically result in knowing approximately how many points will be present in each k-neighborhood. Thus setting the wanted level of detail of each $\mathcal{P}^k$.

Resampling methods are also fundamental in point clouds were the point density is not uniform, for example clouds that are the result of a *Registration* process, where multiple view can be overlapped on the same surface. As an example, the Figure 4.4 shows the result of a registration process with large normals and curvature errors, that gets resampled to smooth these errors.

Many methods of resampling exist in literature, but here we focus on the ones used is our procedural pipeline. As a downsampling procedure we use a *Voxel Grid* filter and for upsampling a *Moving Least Squares* (MLS) with random uniform density, see [BNRV05, ABCO+03] for more informations. The voxel grid filter encapsulate the point cloud in a grid of voxels. Voxels are the "3D" equivalent of pixels in a "2D" image, think about them as tiny boxes with three dimensions, called leaf size. Normally the voxels are set as cubes, so the three dimensions are all equal.

Once the input point cloud has been enclosed in this grid of voxels, for every one of them, the filter searches if there are points inside it, if so it substitutes all of them with just one point located at the centroid of points

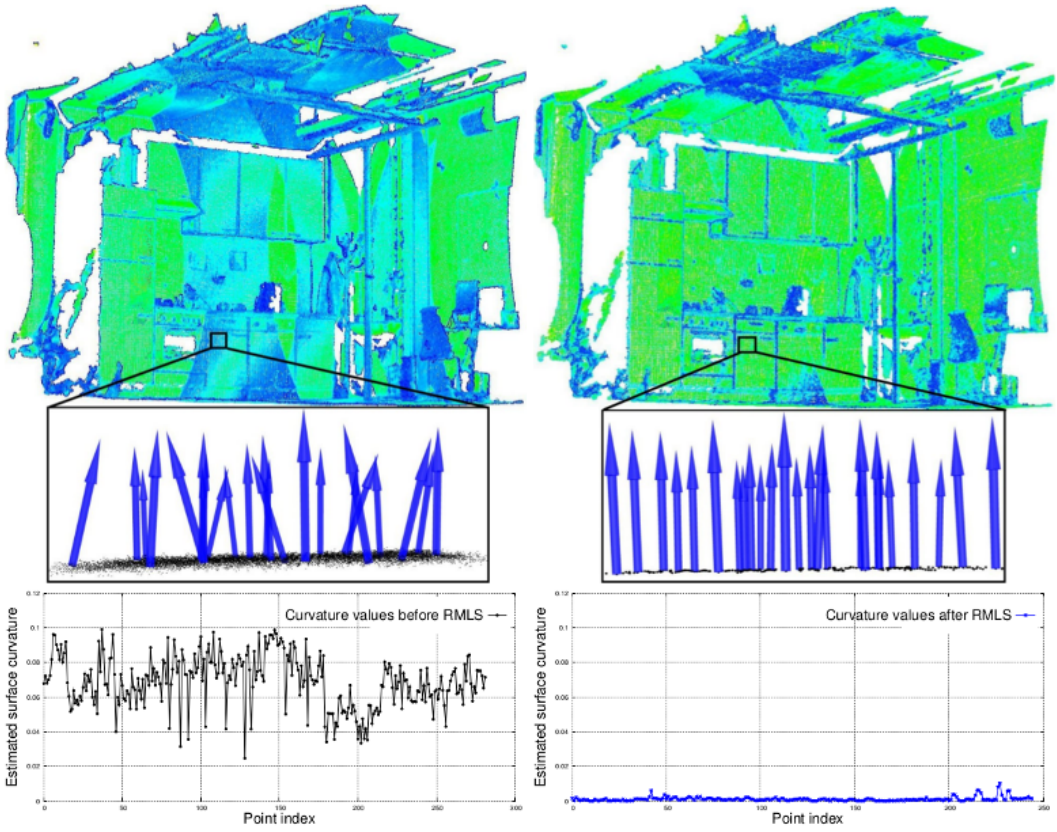


Figure 4.4: Example of a point cloud after registration and its curvature and normal estimation, before resampling (left) and after (right). Note the large curvature and normals imperfections on the point cloud on the left. Picture taken from [Rus10].

enclosed in the voxel. The centroid ($\bar{\mathbf{p}}$) of k points $\mathbf{p}_i$ can be computed with the following:

$$\bar{\mathbf{p}} = \frac{1}{k} \cdot \sum_{i=1}^{k} \mathbf{p}_i \tag{4.3}$$

By adjusting the leaf size one can control the number of points present in the output cloud, thus controlling the filter strength. This approach is a bit slower than approximating the points with the center of the voxel, as other methods do, but it represents the underlying surface more accurately, and thus was preferred for our kind of application.

This filter assures a uniform density of points all along the point cloud, because the resulting points will be approximately spaced “leaf size“ apart. Figure 4.5 illustrates some examples of the Voxel Grid Filter applied to our real database.

The upsampling procedure does the opposite of the previous one, increasing the number of points in the cloud. In particular the Moving Least

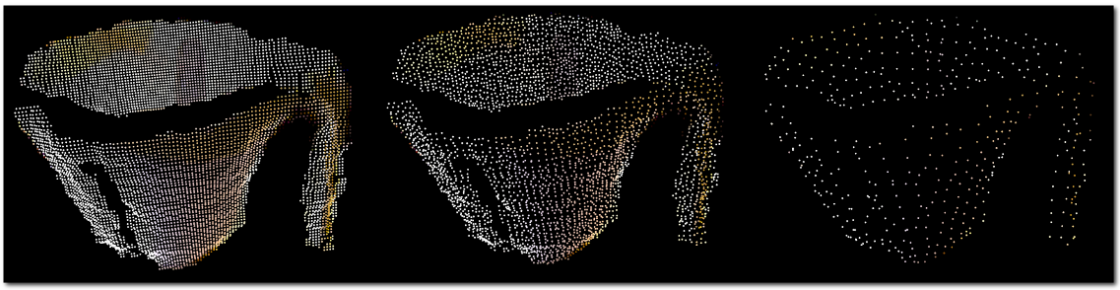


Figure 4.5: Examples of Voxel Grid filter with different leaf sizes: the container on the left is the original point cloud, in the centre the container is downsampled with leaf size of $3mm$, on the right is downsampled with a leaf size of $1cm$.

Squares (MLS) method provides an interpolating surface for a given set of points $\mathcal{P}$ by fitting higher order bivariate polynomials to each point neighborhood locally. Without going into many details, the procedure can be outlined as follows:

1. For every k-neighborhood $\mathcal{P}^k \subseteq \mathcal{P}$ fit a local plane, that approximates the underlaying local surface.
2. Create a set of polynomial functions $f(\cdot)$ of order m so that $f(\mathbf{p}_i) = f_i$.
3. Fit these functions in the set of distances from the points to the surface, so that they minimize the weighted least-square error:

$$\sum_{i \in \mathbb{I}} (a(\mathbf{p}) - f_i)^2 \cdot \Theta(\|\mathbf{p} - \mathbf{p}_i\|) \tag{4.4}$$

   where $a(\mathbf{p})$ is the moving least squares approximation at the point $\mathbf{p}$ and $\Theta(\cdot)$ is the weight function.
4. Project the approximations back to the local surface.

For our data, polynomial functions of order $m = 2$ are more than enough to approximate the local surfaces, because most objects present areas that are either planar or curved in only one direction. The upsampling is performed in between *steps 3-4* by having more approximations $a(\mathbf{p})$ than the original set of points present in the cloud, although other methods for upsampling (like "voxel grid dilation" and "sample local plane") do exists, in this thesis we are more interested in the resampling effect of the method, rather than the effective increase of points.

As visible in Figure 4.6, the result of this procedure is an interpolated surface that is smoother than the original.

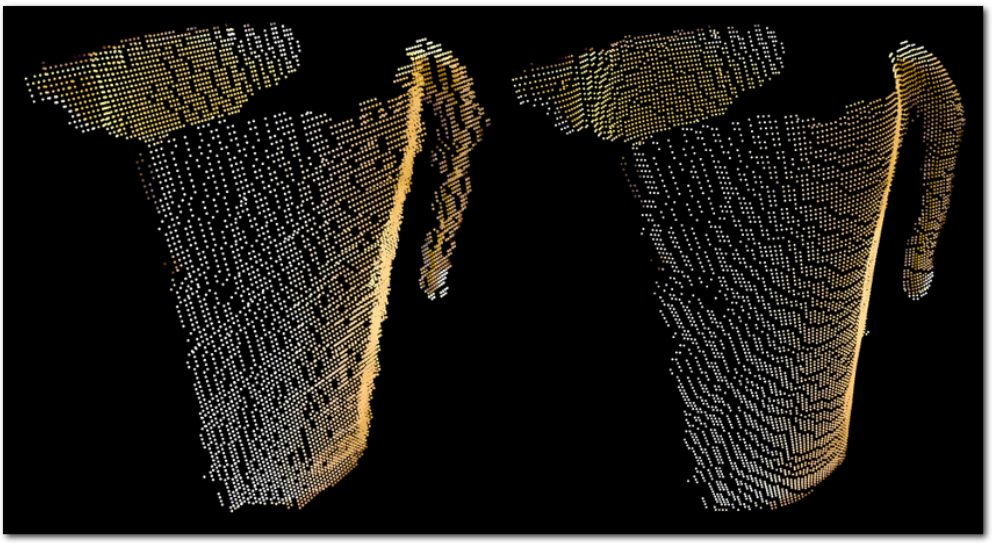


Figure 4.6: Upsample procedure with Moving Least Squares: the original point cloud (left) and the interpolated resulting cloud (right). Note the smoother surface on the output cloud.

This aspect is particularly interesting, because it alleviates the sensor noise. By all means the Moving Least Squares algorithm tries to reconstruct the surface of the cloud, filling small holes and, more importantly to our application, smooths the depth noise of the Xtion sensor. For this reason, the resampling power of the Moving Least Squares was included in or pre-proessing pipeline, to try to improve the general quality of the data at the cost of slightly more computation time.

Sometimes, however, this procedure might introduce outliers along borders or on small surfaces like handles, probably due to the interpolating surface over bordering the object. Anyway to prevent possible introduced noise we decided to reapply a very mild outliers filter to the output of the MLS resampling.

Summarizing, the pre-processing steps we decided to apply to our data, prior the features' estimation, can be listed in this order:

1. Outliers Filtering, with $k = 50$ and $\alpha = 3$. This filter removes spurious outliers that may be left from the acquisition procedure, its parameters are not too restrictive, because our data is not very noisy. Effectively removing points on the borders, to try to smooth them.

2. Moving Least Squares resampling with random uniform density upsampling. The radius r for determing the $\mathcal{P}^k$ was set to $3cm$. The

random uniform density was set to 200 points, meaning that after the fitting, if less than 200 points were found on each $\mathcal{P}^k$, then the procedure introduces new points and fit them on the local MLS surface, so that their total number is 200. Total number of points on each *k-neighborhood* is then spaced and moved on the interpolating surface to achieve uniform density. This may create an upsample or it may not, for our data most of the sub sets of local surfaces didn't require upsampling to reach 200 points.

3. Second Outliers Filter, with $k = 50$ and $\alpha = 4$. The second filter is even milder than the first one and it is aimed at removing outliers that may be introduced by the MLS resampling procedure.

4. Voxel Grid Downsampling, with *leaf size* of $3mm$, for reducing the total number of points to speed up features computation and have a constant density of points along the clouds.

In Figure 4.7 is visible an example of the pre-processing pipeline listed above, applied to a funnel pose, taken from the real database. As one can see the filters removes very few outliers and are more a precaution than a necessity, the MLS resampling instead changes the surface of the object, smoothing it in all directions and partially eliminates the sensor depth noise. Finally the Voxel Grid filter reduces the number of points and roughly put them along the same distance.

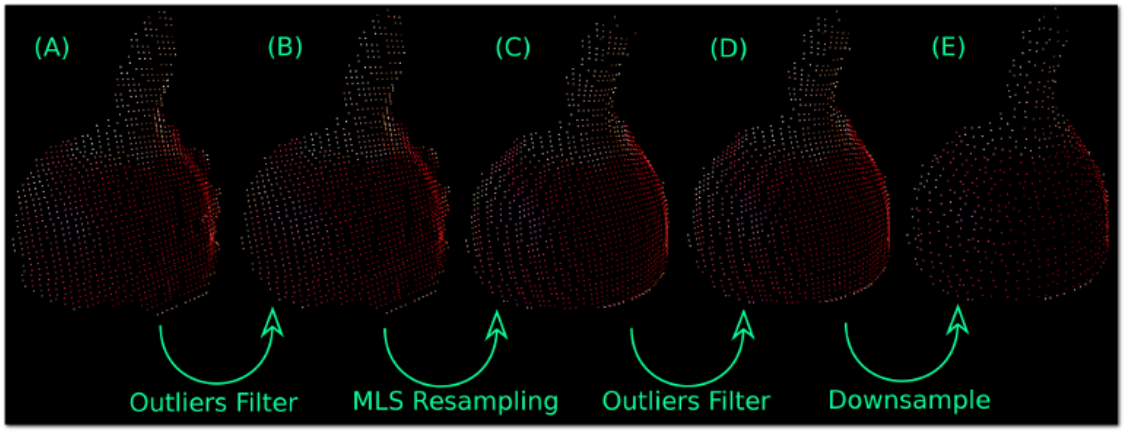


Figure 4.7: Example of our pre-processing pipeline applied to a funnel pose: (A) the acquired object from the *real database*, (B) the object after the first outliers filtering, (C) the result of MLS resampling, (D) the application of the second milder filter, and finally (E) the object after the downsampling with leaf size of $3mm$.

The computational time of the pre-processing steps varies experimentally on the total number of points, so it's different from object to object, but averagely is $700 - 800ms$ on a *AMD FX-6200* six-core processor with $8Gb$ of RAM. Most of the time is occupied by the MLS algorithm (almost 90% of it), so for this reason an alternate faster pipeline was also used. It consists of applying *step 4* directly after *step 1*, thus excluding the slow MLS resampling. The impact on performance of the pose estimation procedure is addressed in Chapters 5 and 6.

Since the outlier filter and voxel grid downsample takes roughly a total of $150ms$ to execute for our data, we decided to apply them almost every time. The outliers filter could be removed to gain $50 - 60ms$ of execution time, but since the gains in time is so little, compared to potential benefits of applying it, we practically never remove it. The downsampling however was never removed from the pre-processing pipeline, because it speeds up the pose estimation procedure considerably, by reducing the number of points in point clouds.

4.3 Normals Estimation

Surface normal estimation is a required passage for almost every "3D" features computation, and it is also applied to our application in effort to estimate the pose of objects. Though many normal estimation methods exist, the one that is explained here is one of the simplest. The problem of determining the normal at a point to a surface is defined by how we chose to approximate this surface.

The method proposed, and used in many applications, is to approximate the local surface with a tangent plane, in the least squares sense, the normal of the query point is then the normal to the tangent plane. Finding this plane, in turn becomes a least-square fitting problem, that can be solved by analyzing the eigenvectors and eigenvalues of a covariance matrix created from the k-neighborhood of the query point. Summarizing, the procedure can be outlined by the following steps:

1. For every point ($\mathbf{p}_i$) in the cloud calculate their k-neighborhoods ($\mathcal{P}_i^k$), defined by a radius r.

2. Calculate the covariance matrix $\mathbf{C}$ for every query point $\mathbf{p}_i \in \mathcal{P}_i$ as:

$$\mathbf{C} = \frac{1}{k} \cdot \sum_{j=1}^{k} \left(\mathbf{p}_j - \bar{\mathbf{p}}\right) \cdot \left(\mathbf{p}_j - \bar{\mathbf{p}}\right)^T \tag{4.5}$$

   where k is the total number of points in the neighborhood of $\mathbf{p}_i$, and $\bar{\mathbf{p}}$ represents the 3D centroid calculated as in (4.3).

3. Calculate the eigenvalues and eigenvectors of $\mathbf{C}$, the normal searched can be approximated by the eigenvector associated to the smallest eigenvalue.

Mathematically the covariance matrix $\mathbf{C}$ is symmetric and positive semi-definite, so its eigenvalues are positive real numbers and consequently the associated eigenvectors form an orthogonal frame. If the eigenvalues are ordered ($0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$) the eigenvector $\vec{\mathbf{v}_0}$ corresponding to the smallest eigenvalue λ_0 is therefore the approximation of the normal we were searching for.

Once the normals of points in the cloud are calculated the procedure terminates. However, since generally there is no mathematical way to solve

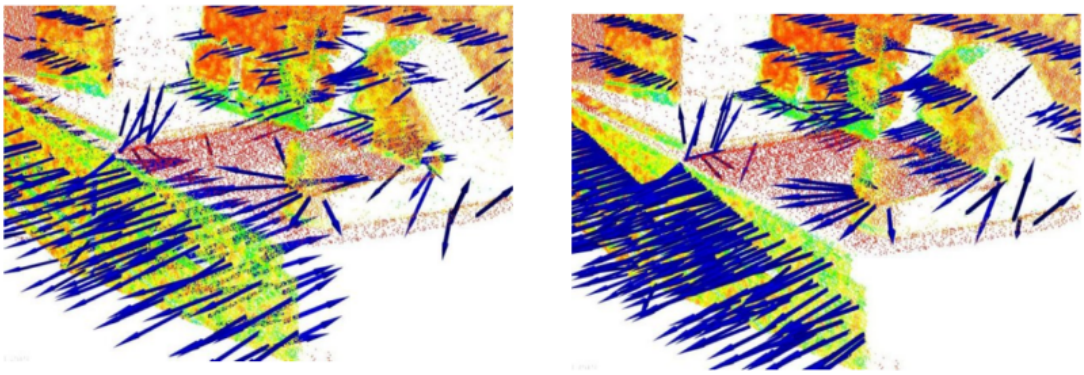


Figure 4.8: Normal estimation without the viewpoint constraint (left) and with it (right). Notice on the right that all the normals are consistently oriented. Image taken from [Rus10].

for the sign of the normal, its orientation computed above is ambiguous, and not consistently oriented over an entire point cloud dataset, as a result the normals can have two opposite orientations. To solve this problem the algorithm makes use of the viewpoint information, the point where the sensor is located during the acquisition, in fact to orient all normals consistently towards the viewpoint, let it be $\vec{\mathbf{v}_p}$, they need to satisfy the following constraint:

$$\vec{\mathbf{n}_i} \cdot (\vec{\mathbf{v}_p} - \mathbf{p}_i) > 0 \tag{4.6}$$

where $\vec{\mathbf{n}_i}$ is the normal at $\mathbf{p}_i$, for this reason the viewpoint information is stored along the point cloud during the acquisition process.

As an example, Figure 4.8 present the estimation of normals for a point cloud, with and without the constraint in (4.6).

From what is explained of this procedure, one can notice that the normal estimation is a kind of local feature, as it was formally defined in Chapter 2. With reference to (2.2) the function $\mathcal{F}\left(\mathbf{p}_i, \mathcal{P}^k\right)$ is in fact the process of normal estimation described. It takes as input a query point and its k-neighborhood and returns the vector of local features, which in this case are the three directions of the normal $\vec{\mathbf{n}_i}$. The features are estimated for all points in the cloud so there's no need, in this case, to compute any keypoints.

This arises the question of computational time, since we discussed that local features are generally slower in computation than their global counterpart. In reality, this is not the case since the calculations required to compute the features can be easily resolved by any modern machine in almost no time. The bottleneck of computation is actually the number of

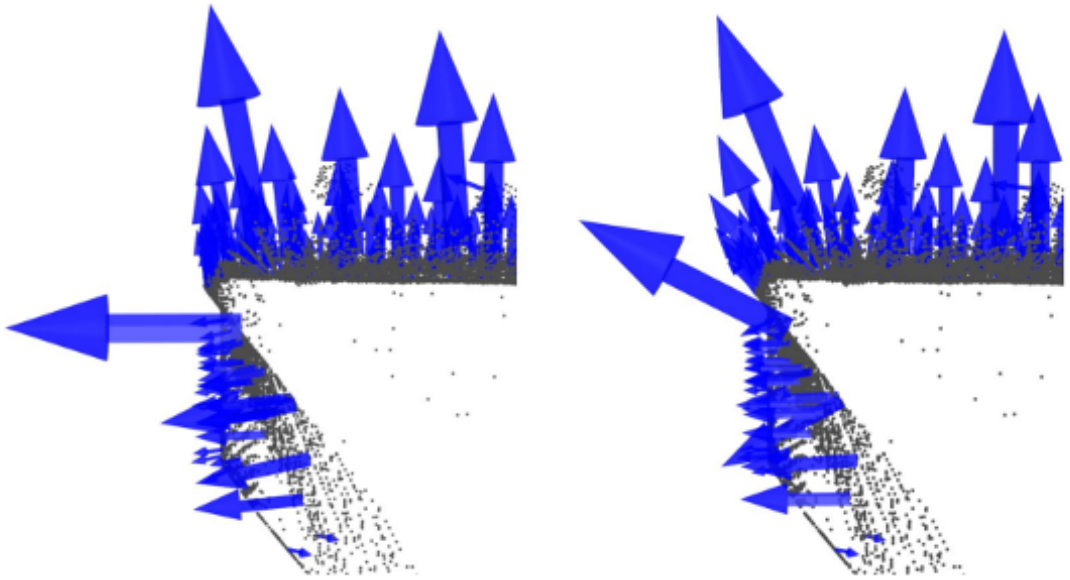


Figure 4.9: Normal estimation in two cases: (left) a reasonably good choice of radius, (right) the radius chosen is too big and the normals get estimated over points from different surfaces, thus modifying the tangent plane. Image taken from [Rus10].

times this procedure needs to be executed, and that is directly proportional to the number of points in the cloud. Considering that large cloud sets may have more than tens of thousands points, the computational time may become an issue.

However with modern multi-core/multi-threaded paradigms, like for example OpenMP [Ope], it is possible to parallelize the executions, speeding up the total computation. For our datasets, which were previously downsampled also for this reason, the mean execution time of the normals' estimation is about $20 - 30ms$, with the use of OpenMP parallelization.

As previously explained, a surface normal at a point needs to be estimated from its $\mathcal{P}^k$. The specifics of the nearest neighbor estimation problem raise the question of the right scale factor: given a sampled point cloud dataset, what are the correct number of neighbors to consider, or the radius to set, that should be used in determining the subset of the nearest neighbors of a point?

This issue is of extreme importance and it limits the full automation of normals' estimation process and consequently the features' extraction, because the user is needed to set these parameters a priori. As said above one method to ease this decision is that of applying a resampling prior to the normal estimation process (Section 4.2), because knowing in advance

the density of points lets the user choose a radius more easily. To better illustrate this issue, the Figure 4.9 presents the effects of selecting a smaller radius versus a larger radius for the subsets $\mathcal{P}_i^k$.

Without going into too many details, it suffices to say that, the radius for the determination of a k-neighborhood has to be selected based on the level of detail required. Simply put, if the curvature at the edge between the handle of a mug and the cylindrical part is important, the radius needs to be small enough to catch those details, and large otherwise.

Experimentally, we found that a good compromise for our dataset, was to have a radius of $1.5cm$ for determining the neighborhoods, also considering the average point density of $3mm$ resulting after the downsample. The results are shown in Figure 4.10.

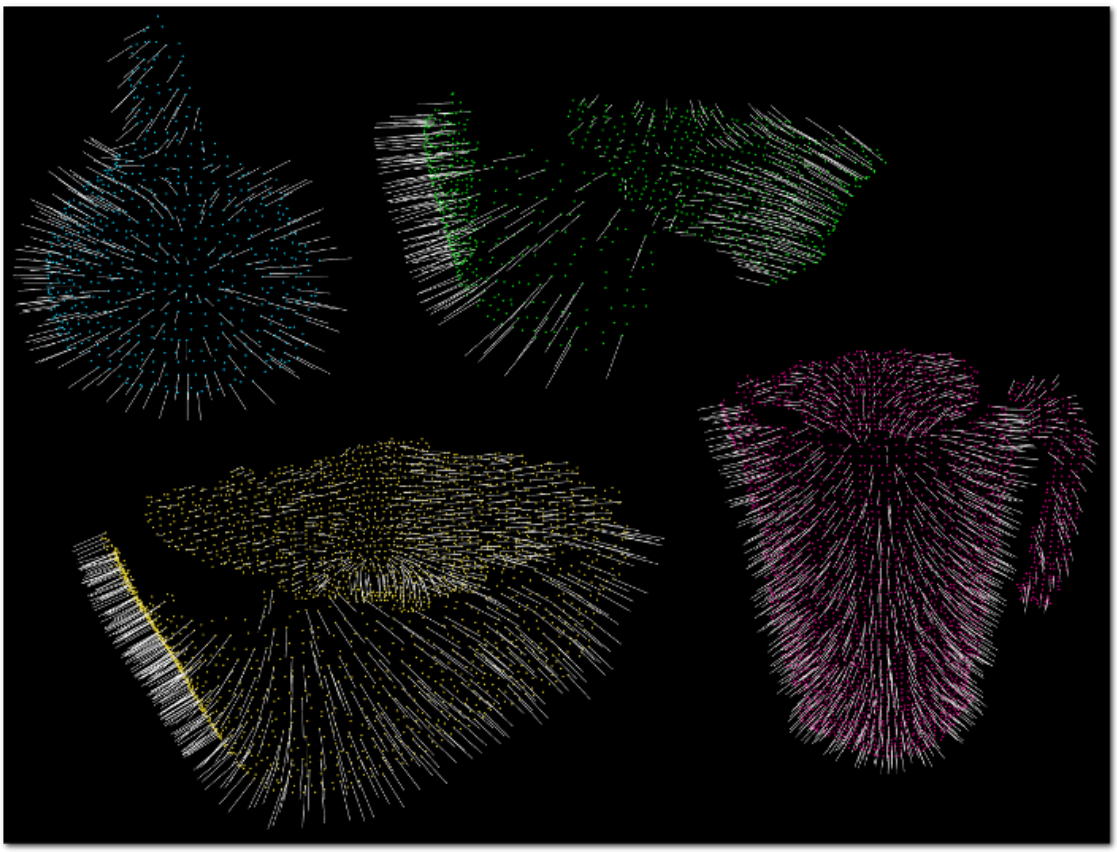


Figure 4.10: Examples of normals estimation from our databases: a funnel (top left), two different containers (top right)(bottom left) and a jug (bottom right). Normals are represented by white lines and are shown only for a thirth of the total points to avoid confusion.

Global Feature Estimation

5

"Science is nothing but perception"

Plato (428BC - 348BC)

This chapter presents a panoramic of the methods for computing *global features* from a point cloud. As first introduced in Section 2.4 we defined a global feature as some function that maps the points in a cloud to some n-dimensional space (2.4). The nature of this $\mathcal{G}(\cdot)$ function, or $\mathcal{F}(\cdot)$ for the local variant, is what defines the features themselves.

Quite a few methods for defining this functions do exists, the Table 5.1 shows the most common. The table groups the features as local or global and those used in this thesis are highlighted in red and discussed in the next sections.

There is no better approach or method, everyone has pros and cons, and most likely the choice of a method over another should be done depending of the type of application and the performance required. In this thesis we decided to not take this choice, instead we used all the four global features and tried to merge their results in effort to take advantage from all of them. This "combination" of features will be discussed in Chapter 6. We did not include the global variant of Point Feature Histogram or PFH, because it is one the slowest feature to calculate (more than $1s$ for our data) and did not fit well along the others.

In principle a feature, be it global or local, should distinguish itself by being able to capture the various surface characteristics in presence of:

- **Noise** - the feature representation should retain the same or very similar values in the presence of mild noise in the data.
- **Rigid Transformations** - rotations and translations in the data should not influence the resultant feature vector.

- **Varying Density** - the surfaces, sampled with more or less dense points, should have the same feature vector signature.
- **Scale** - the surfaces should retain the same feature vector in presence of scale transformations.

The four global features, used in the thesis, have most the proprieties listed above. They are: the Viewpoint Feature Histogram or VFH, discussed in Section 5.1, the Clustered Viewpoint Feature Histogram or CVFH, discussed in Section 5.2, the Ensemble of Shape Functions or ESF, discussed in Section 5.3, and the Oriented Unique and Repeatable Clustered Viewpoint Feature Histogram or OUR-CVFH, discussed in Section 5.4. All the features listed in Table 5.1, except the ESF, make use of the local normals' estimation, thus they need to be estimated a priori.

Finally, the chapter presents some test we made, in effort to evaluate the object recognition of the various features proposed; those are visible in Section 5.5.

Table 5.1: Methods for estimating 3D features

Classification of Features	
Global Features	**Local Features**
PFH	PFH
VFH	FPFH
CVFH [a]	SC
OUR-CVFH [a]	RSD
ESF [b]	USC
	SIs
	SHOT

a. CVFH and OUR-CVFH are classified as global, but in a sense they are hybrid, because depending of the geometry of the object, they may output more than one feature per cloud.

b. ESF is a particular method that doesn't require a prior normals' estimation, so it can be computed directly after the pre-processing steps.

• Methods highlighted in red represents those used within the thesis.

5.1 Viewpoint Feature Histogram (VFH)

The Viewpoint Feature Histogram is a global feature that maps the whole point cloud into a signature vector of 308 elements. Its computation is introduced in the original papar [RBTH10] and it's only briefly summarized here, for more informations we address the reader to view the cited paper. Specifically the resulting feature vector is a histogram and it is composed by the union of three smaller histograms:

- A histogram composed from the relation between the viewpoint direction and each point normal, totalling 128 bins.
- A histogram obtained from the relation of surface angles of the object, 135 bins.
- A histogram calculated from the distance between the centroid and the points, another 45 bins.

To compute the first part, a histogram of the angles that the viewpoint direction makes with each normal is collected. By this we do not mean the view angle to each normal as this would not be scale invariant, but instead we mean the angle between the central viewpoint direction translated to each normal. More specifically the following steps are performed:

1. Compute the point cloud centroid, resulted from averaging the coordinates of all points (n), be it:

$$\bar{\mathbf{c}} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{p}_i \tag{5.1}$$

2. Compute the vector between this centroid and the viewpoint (the position of the sensor) and normalize it, let's call this vector the *central viewpoint direction* or $\vec{\mathbf{n}_c}$.
3. Iterate over all points and translate to each of them the central viewpoint direction, then evaluate the angle between this vector and the normal at the point (β).
4. Bin the values of β into a histogram of 128 bins.

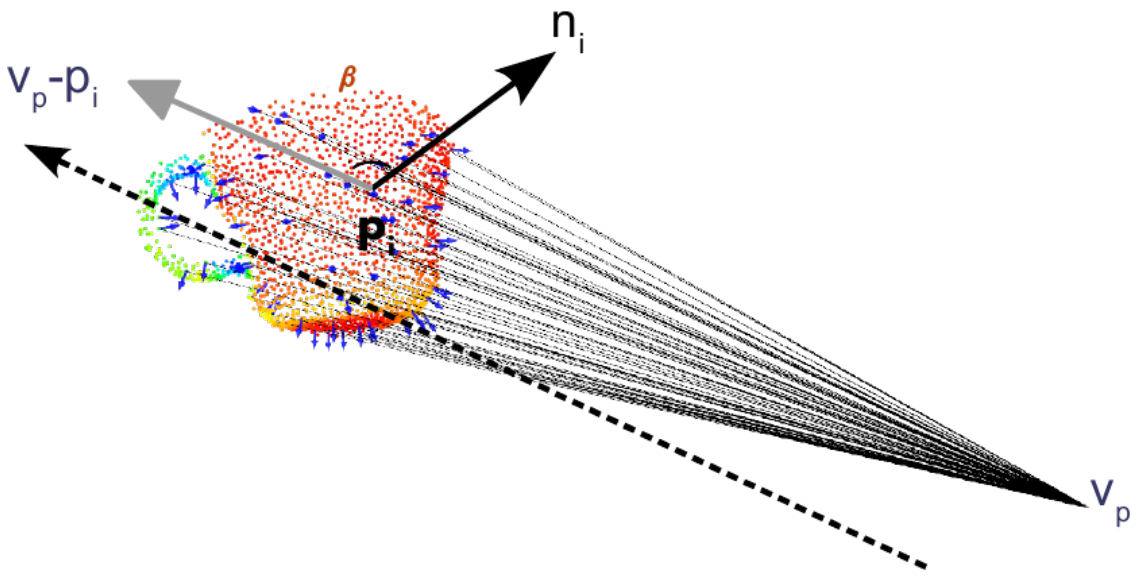


Figure 5.1: Illustration of angle evaluations between the central viewpoint direction and the normal at each point. The central viewpoint direction (dotted line) gets translated to each point and the angle (β) evaluated. The image is taken from the original paper [RBTH10].

Figure 5.1 illustrates this process.

The second component of the VFH feature measures the relative pan, tilt and yaw angles between the central viewpoint direction and each normal. These set of angles are evaluated from a *Darboux frame* coordinate system at the cloud centroid. A Darboux coordinates system $\{\vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}\}$ between a pair of points, in this case $\bar{\mathbf{c}}$ and $\mathbf{p}_i$ is defined as follows:

- Set $\vec{\mathbf{u}}$ as the normal at the point. Since for the VFH estimation, this reference frame is computed at the cloud centroid and it doesn't have a normal, set it as the central viewpoint direction.

$$\vec{\mathbf{u}} = \vec{\mathbf{n}_c}$$

- Set $\vec{\mathbf{v}}$ as the cross-product between the distance vector of the point pair and the first axis.

$$\vec{\mathbf{v}} = (\mathbf{p}_i - \bar{\mathbf{c}}) \times \vec{\mathbf{u}}$$

- Set the last axis as the cross-product between the other two.

$$\vec{\mathbf{w}} = \vec{\mathbf{u}} \times \vec{\mathbf{v}}$$

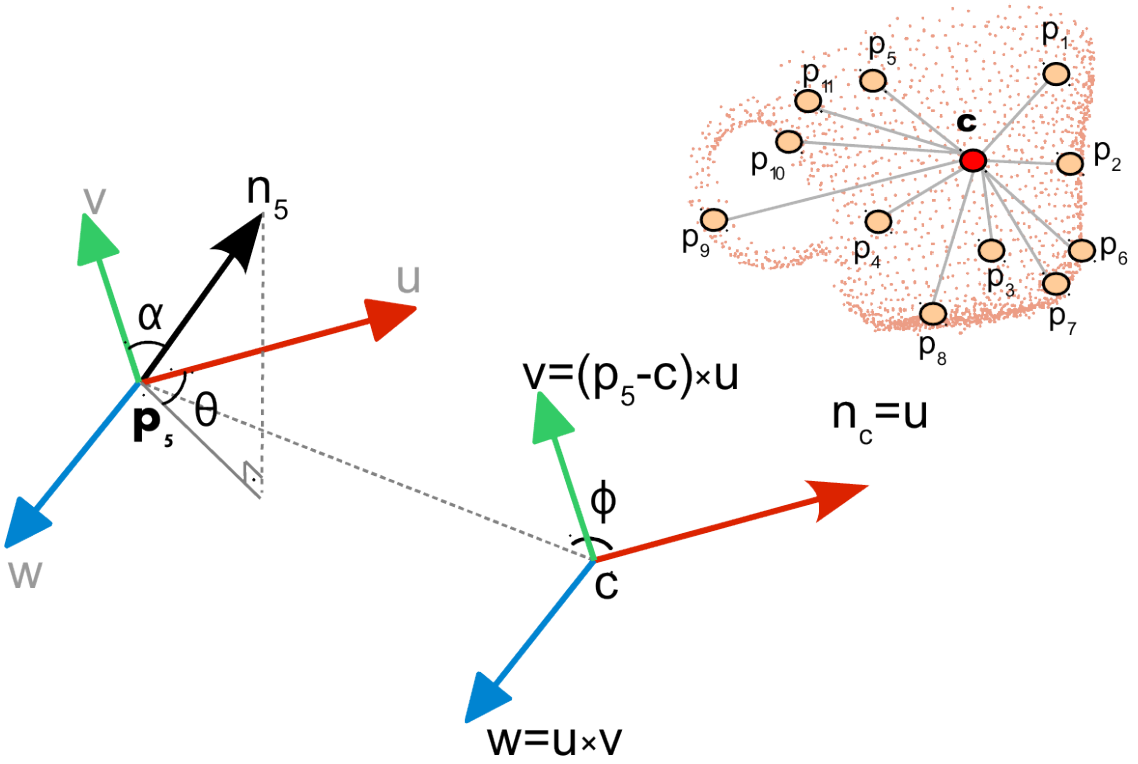


Figure 5.2: Angular component of the VFH estimation: some point pairs selected from the cloud (top-right), a Darboux frame system and the angles computed for an example point pair (bottom-left). The image is taken from the original paper [RBTH10].

With reference to Figure 5.2(bottom-left) the three angles α, ϕ and θ, representing pan, tilt and yaw, can be computed with the following:

$$\begin{aligned} \alpha &= \vec{\mathbf{v}} \cdot \vec{\mathbf{n}_i} \\ \phi &= \vec{\mathbf{u}} \cdot \frac{(\mathbf{p}_i - \bar{\mathbf{c}})}{\|\mathbf{p}_i - \bar{\mathbf{c}}\|} \\ \theta &= arctan(\vec{\mathbf{w}} \cdot \vec{\mathbf{n}_i}, \vec{\mathbf{u}} \cdot \vec{\mathbf{n}_c}) \end{aligned} \tag{5.2}$$

The angles are computed for every point pair between the cloud centroid ($\bar{\mathbf{c}}$) and every other point in the cloud. The values of α, ϕ and θ are collected into a histogram of 45 bins each, thus totalling 135 bins. Figure 5.2(top-right) shows how the point pairs are selected in a given cloud.

The last histogram that compose the VFH feature is called *Shape Distribution Component* ($\mathcal{SDC}$) and it is computed as follows:

$$\mathcal{SDC} = \frac{(\bar{\mathbf{c}} - \mathbf{p}_i)^2}{\max\limits_{\mathbf{p}_i} \left[(\bar{\mathbf{c}} - \mathbf{p}_i)^2 \right]} \tag{5.3}$$

where $\bar{\mathbf{c}}$ is again the centroid of the cloud and $\mathbf{p}_i$ are the other points. These values gets binned into a 45 bins histograms, thus totaling a size of

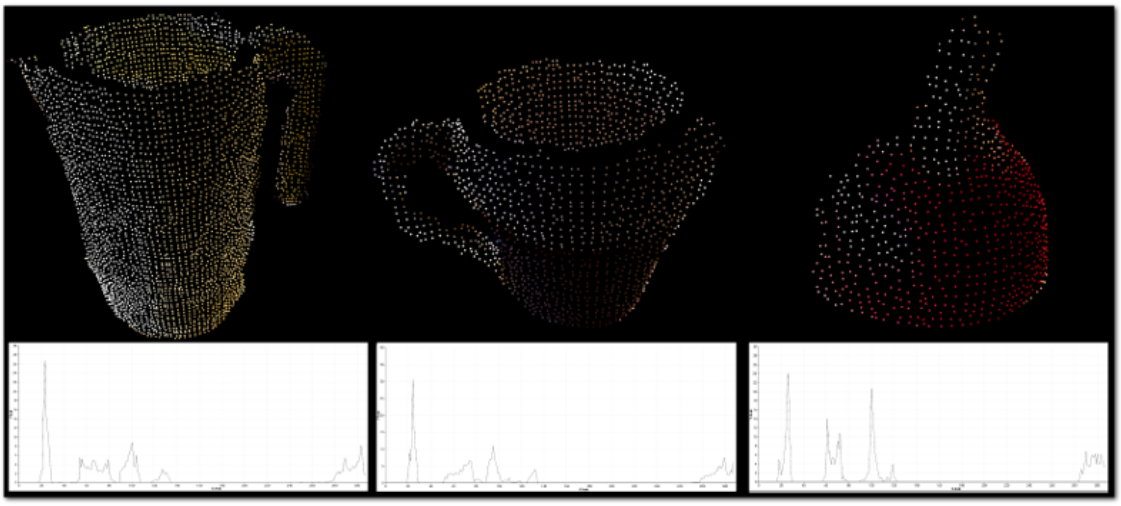


Figure 5.3: VFH histograms of different objects: a jug (left), a mug (centre) and a funnel (right). Note the difference in the histograms peaks, in magnitude and in position.

308 for VFH. This last component was later introduced in [AVB+11] and allows differentiating surfaces that have very similar normals and size, but have their points distributed differently. For example it can distinguish an elongated plane from a more compact one.

The global VFH feature is then concatenated from the previous histograms with the following:

$$\mathbf{VFH} = (\alpha, \phi, \theta, \mathcal{SDC}, \beta) \tag{5.4}$$

As reported in the paper [RBTH10], the computational complexity of VFH is $\mathcal{O}(n)$, thus linear in the total number of points of the cloud. Experimentally, for our databases, the computation of VFH takes averagely less than $0.5ms$ on our *AMD FX-6200* six-core machine, and this fast computation is one of the reasons we chose this feature, along with the others.

In Figure 5.3, it is presented the VFH histograms of sample objects taken from the real database, as one can see, the histograms are quite different from each other, because, as expected, they represent different objects. Only the first two are a little more similar, this is most likely because the objects they represent, both present two cylindrical surfaces, a convex and a concave one. So, in a way, they are more similar between each other than the third object, that doesn't present those cylindrical surfaces.

5.2 Clustered Viewpoint Feature Histogram (CVFH)

The Clustered Viewpoint Feature Histogram is an extension of VFH and it's classified as a hybrid or regional method, because it can create one or more histograms from a given object point cloud, depending on its geometry.

It was first introduced in [AVB+11] as a mean to perform recognition and pose estimation in presence of partially occluded objects and mild noise. In fact one limitation of the VFH feature, presented in the previous section, is the inability to recognize partially occluded objects. Since it would treat them as different, because the underlying geometry of the partial object would result diverse from the complete one, thus leading to different features.

The idea of the CVFH is very simple: subdivide the objects into stable clusters and then compute the VFH for each of them. This way even if the object is occluded but one of its clusters is still visible, it is possible to match it with a cluster of the non covered object, thus performing a recognition. In summary the CVFH computation does the following steps:

1. Subdivide the point cloud into clusters with a smooth regional growing algorithm.
2. Estimate the VFH feature for each cluster.
3. The collection of VFH features represent the global CVFH.

To achieve the stable regions clustering (*step 1*) the following is performed.

First points with high curvature are removed from the cloud, these points, if present, most likely indicate the presence of an edge or high noise in the sensor. The remaining points are clustered with a smooth region growing algorithm.

Each new cluster is initialized with a random point, then a point $\mathbf{p}_i$ with normal $\vec{\mathbf{n}_i}$ is added to the cluster $\mathcal{C}_m$ if the cluster contains another point $\mathbf{p}_j$ with normal $\vec{\mathbf{n}_j}$ in the k-neighborhood of $\mathbf{p}_i$ with similar normal. In detail, the following constraint is fulfilled:

$$\exists \mathbf{p}_j \in \mathcal{C}_m : \|\mathbf{p}_i - \mathbf{p}_j\| < r \wedge \vec{\mathbf{n}_i} \cdot \vec{\mathbf{n}_j} > t_n \tag{5.5}$$

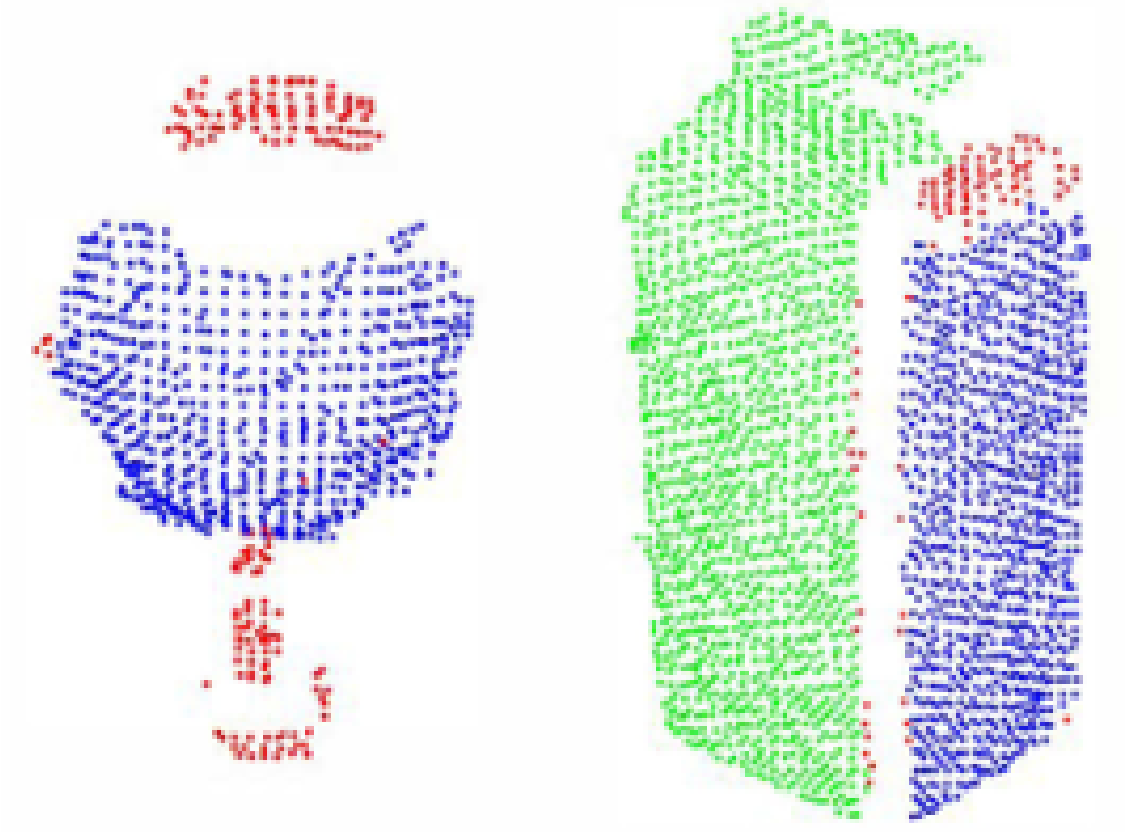


Figure 5.4: Examples of the smooth region segmentation algorithm during the CVFH estimation. Green and blue points represent stable clusters, while red ones don't belong to any cluster. Points with high curvature are not shown, for example the ones on the edge of the milk cartoon. Image taken as it is from the original paper [AVB+11].

where r is the chosen distance between neighboring points, i.e. the radius of the k-neighborhood of $\mathbf{p}_i$. While t_n is a threshold that defines normals similarity. In addiction each cluster must contain at least 50 points in total to be considered stable and thus taken into account. It is possible that some points may end up not belonging to any clusters, or they form up a non-stable one and thus they are ignored for the remaining computations.

Figure 5.4 shows some examples of the smooth region segmentation algorithm, for further details the reader is remanded to the original paper [AVB+11]. The final steps of CVFH computation consist of estimating the VFH (see Section 5.1) of each stable region and list them all in a vector, that globally defines the CVFH feature:

$$\mathbf{CVFH} = (\mathbf{VFH}_1, \mathbf{VFH}_2, \cdots \mathbf{VFH}_m) \tag{5.6}$$

Since this segmentation process is the core of the whole CVFH estimation, the parameters in (5.5) play a fundamental role in the feature definition,

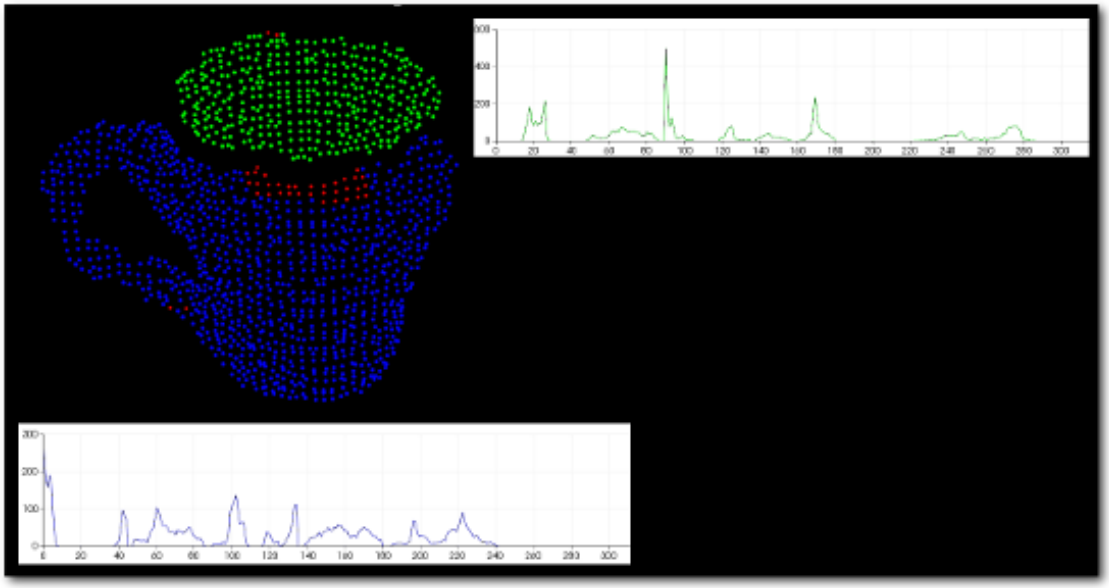


Figure 5.5: CVFH estimation of a mug from the real database: green points represent the first cluster with the corresponding VFH histogram next to them. Blue points represent the second cluster, with the corresponding histogram underneath. Red points do not belong to any cluster, because their normals differ from the ones of their neighbours, more than the angle threshold we set or were removed previously due to high curvature. Plus they are not enough to form another cluster on their own (a minimum of 50 points was set).

because they define the number and quality of the clusters found. By taking into account the suggestions from the authors and the variety of our data, the r parameter was set to $1cm$, circa the triple point density, and the t_n parameter to $cos(7.5°)$.

Figure 5.5 shows a mug from our real database exposed to CVFH estimation, as one can see, two stable clusters are found and for each of them a VFH histogram is computed.

It is important to note that, even if only one cluster is found on the cloud, it is not the same as computing the VFH from the input cloud; in fact the regional segmentation excludes points on sharp edges and in non-stable clusters. For example, for the glass on the left of Figure 5.4 only one stable region is found (blue), but the VFH is computed for those points only and not for all the cloud. This makes the two features (VFH and CVFH) distinct and so, in our application, they were both computed and treated as different.

In summary the CVFH feature is the union of VFH, calculated from stable clusters, and its cardinality m depends on the object geometry. It is

possible that even the same object could have different clusters if viewed from different viewpoints, rendering the matching of objects slightly more difficult. A method to efficiently compare objects with CVFH (and also OUR-CVFH, since it presents the same problem, see Section 5.4) was created and it will be discussed in Section 5.5.

Computationally the CVFH is more cumbersome to compute than VFH, but it is still very fast; experimentally $20 - 30ms$ were more than enough to estimate the feature for most of our data.

One disadvantage of CVFH (and also VFH) is the inability to distinguish between roll rotations around the viewpoint direction, i.e. rolling an object or the camera will produce the same exact feature. This problem leaves these features unable to estimate a full 6 DoF pose of a given object.

For this reason the authors of CVFH introduced another histogram, to be paired with CVFH, that could measure the variance introduced by the roll angle. They called it Camera Roll Histogram (CRH) and measured the full 6 DoF pose from the CVFH, CRH pairs.

However, in our pipeline for pose estimation, we don't use CRH, since we use all these features as a mean to recognize objects and the pose is estimated with different algorithms, see Chapter 6.

5.3 Ensemble of Shape Functions (ESF)

The Ensemble of Shape Functions is a global feature, that is an ensemble of ten 64 bin histograms of shape functions, describing characteristic properties of the point cloud. The ensemble histogram, first introduced in [WV11], is then composed by 640 bins.

The shape functions describe angles, point distances and area shapes. They all need just the point position information in the cloud, so this feature doesn't need the estimation of normals to the surface and can be calculated directly from the sensor acquisition.

The procedure to estimate the feature can be summarized with the following:

1. Enclose the input point cloud into a voxel grid as an approximation of the real surface, to improve computation time and to separate the shape functions into more descriptive histograms.
2. Iterate over all points in the voxel grid and for every iteration, sample three random points $\mathbf{p}_i$, $\mathbf{p}_j$, $\mathbf{p}_k$ and calculate the shape functions for them.
3. Bin the values of the shape functions into ten 64 bin histograms.
4. Assemble the ESF feature from the ten histograms created above.

The shape functions used are four:

- *D2*
- *D2 Ratio*
- *D3*
- *A3*

The *D2* function calculates distances between a pair of points, since there are three random points in every iteration, the function is evaluated three times, one for every possible point pair out of three.

Suppose the two points in exam are $\mathbf{p}_i$ and $\mathbf{p}_j$, the values calculated from the function are binned into three distinct 64 bin histograms, according to the following rules:

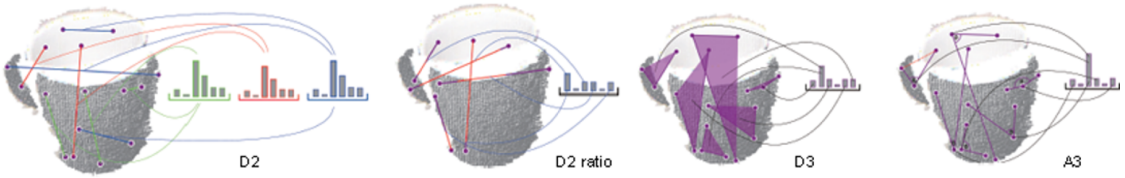


Figure 5.6: Illustration of ESF shape functions. *D2* shape function and corresponding histograms (left), *D2 Ratio* and corresponding histogram (centre-left), *D3* function (centre-right) and *A3* (rigth). Note that, even if *D3* and *A3* have only one histogram shown in the Figure, they have three each, like *D2*. Figure is taken from the original paper [WV11].

- If the line connecting $\mathbf{p}_i$ and $\mathbf{p}_j$ lays entirely on the surface, i.e. there's a surface on the object that covers the entire line, then bin the value of $\|\mathbf{p}_i - \mathbf{p}_j\|$ into the $D2_{IN}$ histogram.
- If the line connecting $\mathbf{p}_i$ and $\mathbf{p}_j$ is completely outside the surface, i.e. no part of the line lies on any part of the object, then bin the value of $\|\mathbf{p}_i - \mathbf{p}_j\|$ into the $D2_{OUT}$ histogram.
- If the line connecting $\mathbf{p}_i$ and $\mathbf{p}_j$ is a mix of the two previous cases, i.e. the line intersect the object, then bin the value of $\|\mathbf{p}_i - \mathbf{p}_j\|$ into the $D2_{MIXED}$ histogram.

The *D2 Ratio* function captures, for every line described by two points pair, the ratio between parts of the line lying on the object and in free space. In other words, how much of the line is on the object surface and how much isn't. This ratio should be 0 for entirely outside and 1 for entirely inside, then lines classified in $D2_{MIXED}$ have values between these extremes.

This value then increment the corresponding bin in another 64 bin histogram called $D2_{RATIO}$.

The *D3* function calculates the square root of the area of the triangle shaped by the three random points $\mathbf{p}_i$, $\mathbf{p}_j$ and $\mathbf{p}_k$, then the value computed is binned into three histograms following the same rules as for the *D2* function:

- If the area completely covers the object, bin it into the $D3_{IN}$ histograms.
- If the area doesn't cover any part of the object, then bin it into the $D3_{OUT}$ histogram.
- Finally if the area covers a part of the object, but not all of it, then bin it into the $D3_{MIXED}$ histogram.

For the last shape function ($A3$) the angle between the three points is calculated, for example the angle in $\mathbf{p}_i$, shaped by the two other segments $\overline{\mathbf{p}_i\mathbf{p}_j}$ and $\overline{\mathbf{p}_i\mathbf{p}_k}$. Thus the angle $\widehat{\mathbf{p}_j\mathbf{p}_i\mathbf{p}_k}$. The values of these angles are binned into another three 64 bin histograms similarly as the others:

- If the line opposite the angle lies entirely on the object, then bin the angle into $A3_{IN}$. In the above example this segment is $\overline{\mathbf{p}_j\mathbf{p}_k}$.
- If the line opposite the angle is completely outside the object, bin the angular value into $A3_{OUT}$.
- If the opposite line lies on the object and also outside it, bin the angle into $A3_{MIXED}$.

The four shape functions are visible in Figure 5.6.

Summarizing, after the shape functions are calculated for every point taken in groups of three, ten histograms of 64 bins are filled. The ESF feature is the union of these histograms:

$$\begin{aligned}\mathbf{ESF} = \big(&D2_{IN}, D2_{OUT}, D2_{MIXED}, D2_{RATIO}, D3_{IN}, D3_{OUT}, D3_{MIXED},\\ &A3_{IN}, A3_{OUT}, A3_{MIXED}\big)\end{aligned} \tag{5.7}$$

so the final histogram has 640 bins.

Experimentally the computational time of ESF takes about $50-60ms$, rendering it one of the slowest global feature to calculate, but still reliably fast in the economy of the pose estimation pipeline.

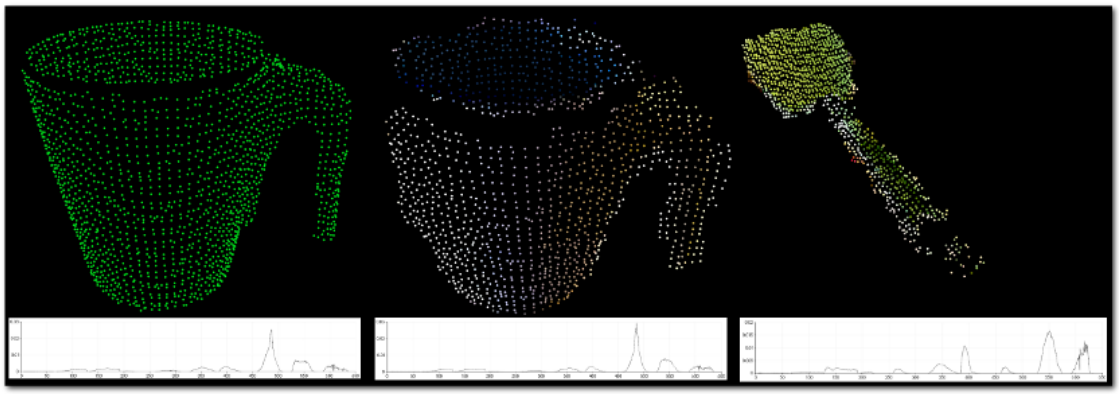


Figure 5.7: ESF features of sample object poses: a mug from the synthetic database (left), the same mug from the real database (centre) and a spoon from real database (right). Note the similarity between the histograms of the first two objects.

Finally, Figure 5.7 presents a few objects belonging to the real and synthetic database, with the ESF feature estimated. The first two objects represent the same mug taken from the same pose, but the first acquired with the synthetic scanner, the second with the real one.

As one can see the histograms underneath them are quite similar, as expected the two point clouds representing the same object gets mapped with a similar feature, leading to a probable match in the feature space.

The third object, on the other hand, is very different from the others and in fact its feature also is different.

5.4 Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram (OUR-CVFH)

The Oriented, Unique and Repeatable CVFH expands the previous feature, see Section 5.2, by adding the computation of a unique reference frame to make it more robust. OUR-CVFH was first introduced in [ATRV12] as mean to improve the spatial descriptiveness of CVFH and remove the invariance to camera roll, while still maintain all the advantage of the previous feature.

To accomplish this, the procedure makes use of *Semi-Global Unique Reference Frames* (SGURFs), which are repeatable coordinate systems computed for each region, or clusters.

The first part of the computation is akin to CVFH, but with some variants. After this a SGURF is estimated for each cluster and the final OUR-CVFH is assembled, similarly as on CVFH. The feature computation can be summarized with the following steps:

1. Subdivide the point cloud into clusters with smooth regional growing, and refine the clusters obtained.
2. Estimate a SGURF for each cluster.
3. Compute a variant of VFH for each cluster.
4. Assemble OUR-CVFH as a collection of the previous VFHs.

During *step 1*, the clustering process follows its predecessor, but differently to CVFH, every point in the cluster is filtered once more by the angle between the normal at the point and the average normal of cluster points. This results in better shaped clusters for a more robust estimation of the reference frame directions. Figure 5.8 shows the clusters of different surfaces before and after the filtering stage.

Figure 5.8: OUR-CVFH and CVFH clusters for different objects, left and right respectively. Figure is taken from original paper [ATRV12].

For every cluster obtained this way, *step 2* takes place. The computation of SGURF for a cluster $\mathcal{C}_i$, whose calculated centroid is $\bar{\mathbf{p}}$ and averaged normal is $\vec{\mathbf{n}}$, is as follows:

(A) Set origin of SGURF in $\bar{\mathbf{p}}$, then compute the eigenvectors of the weighted scatter matrix of points in the cluster:

$$\mathbf{M} = \frac{1}{\sum_{j \in \mathcal{C}_i} \left(R - d_j\right)} \sum_{j \in \mathcal{C}_i} \left(R - d_j\right)\left(\mathbf{p}_j - \bar{\mathbf{p}}\right)\left(\mathbf{p}_j - \bar{\mathbf{p}}\right)^T \tag{5.8}$$

where $d_j = \|\mathbf{p}_j - \bar{\mathbf{p}}\|$ and $R = max\left[d_j, \forall j \in \mathcal{C}_i\right]$.

(B) The $\vec{\mathbf{z}}$ of SGURF is assigned as the eigenvector associated to the smallest eigenvalue of $\mathbf{M}$, the sign of the axis is disambiguated by taking the one yielding a positive dot-product with $\vec{\mathbf{n}}$.

(C) At this point, one among the remaining eigenvectors $(\mathbf{v}_1, \mathbf{v}_2)$ is chosen as $\vec{\mathbf{x}}$ for SGURF. The method for choosing which one and also to disambiguate its sign is carried out by evaluating the difference of point density between the two hemispheres defined by each eigenvector. For instance, let $\mathbf{v}_1^-$ be the opposite vector of $\mathbf{v}_1$, then its sign is disambiguated as follows:

$$\begin{aligned} S^+_{\mathbf{v}_1} &= \sum_{j \in \mathcal{P}} \|\left(\mathbf{p}_j - \bar{\mathbf{p}}\right) \cdot \mathbf{v}_1\| \cdot \left[\left(\mathbf{p}_j - \bar{\mathbf{p}}\right) \cdot \mathbf{v}_1 \geq 0\right] \\ S^-_{\mathbf{v}_1} &= \sum_{j \in \mathcal{P}} \|\left(\mathbf{p}_j - \bar{\mathbf{p}}\right) \cdot \mathbf{v}_1\| \cdot \left[\left(\mathbf{p}_j - \bar{\mathbf{p}}\right) \cdot \mathbf{v}_1^- > 0\right] \\ \mathbf{v}_1 &= \begin{cases} \mathbf{v}_1 & \text{if } |S^+_{\mathbf{v}_1}| \geq |S^-_{\mathbf{v}_1}| \\ \mathbf{v}_1^- & \text{otherwise} \end{cases} \end{aligned} \tag{5.9}$$

The sign disambiguation is performed also for the other vector and is done by taking into account all the points in the cloud ($\mathcal{P}$) and not just the cluster points. However the centroid $\bar{\mathbf{p}}$ is still the one relative to the cluster.

This characterizes the global aspect of SGURF, hence its adjective (*Semi-Global*). Finally to chose which eigenvector will be $\vec{\mathbf{x}}$ of SGURF a

disambiguation factor (f) is computed for both:

$$f_i = \frac{min\left(|S^-_{\mathbf{v}_i}|, |S^+_{\mathbf{v}_i}|\right)}{max\left(|S^-_{\mathbf{v}_i}|, |S^+_{\mathbf{v}_i}|\right)}, i = 1, 2 \tag{5.10}$$

The factor ranges in $[0,1]$, where 0 represent perfect disambiguation while 1 complete ambiguity. The one with lower disambiguation factor is choses as $\vec{\mathbf{x}}$ for SGURF.

(D) the final $\vec{\mathbf{y}}$ is chosen as $\vec{\mathbf{x}} \times \vec{\mathbf{z}}$.

Unfortunately, in some cases the disambiguation for SGURF is not robust, this can happen when both disambiguation factors (5.10) are similar. As a result two reference frames need to be generated, one for each eigenvector.

The worst case, however, happens when both disambiguation factors are close to 1, if this happens also the sign disambiguation (5.9) fails, because both $S^+_{\mathbf{v}_i}$ and $S^-_{\mathbf{v}_i}$ are similar. To account this case, four reference frames ought to be generated, including both eigenvectors, each encompassing both signs.

Figure 5.9(left) shows an object with a stable cluster and the relative SGURF.

When all the SGURFs are computed, the algorithm starts estimating the VFHs of the smooth clusters (*step 3*), but with some variants respect to the previous methods. In reference to (5.4), the α, ϕ, θ and β components of

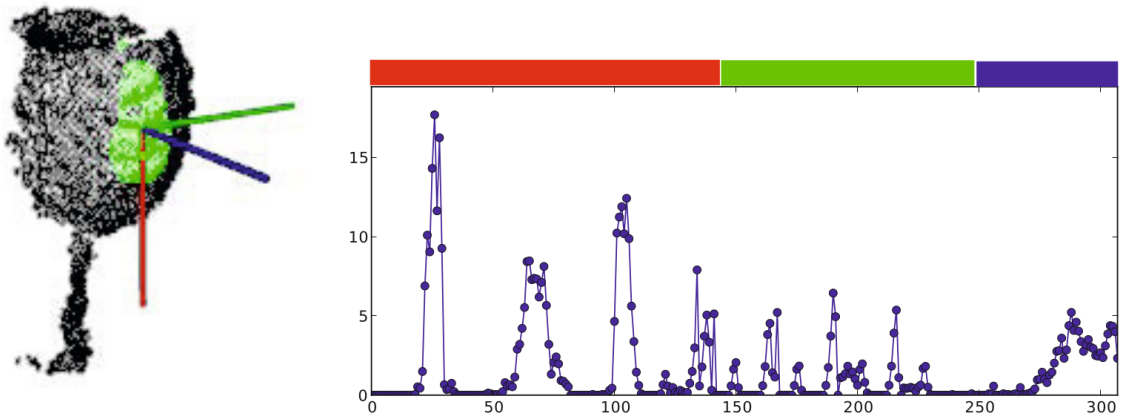


Figure 5.9: On the left a point cloud of a glass with a cluster (green) and its associated SGURF. On the right the resulting OUR-CVFH histogram. Red and blue bins represent the normal distributions and viewpoint component of VFH. Green bins represent the 8 spatial distributions obtained from the points in each octant. Figure is taken from original paper [ATRV12].

VFH remains the same, with β changed to 64 bins instead of 128. Since normals are always pointing towards the viewpoint, their dot-product with the central view direction is ensured to be in the range $[0,1]$ and so there is no need to reserve histogram space for the rest of the range.

The $\mathcal{SDC}$ component instead is completely removed and replaced with a spatial description calculated with SGURF. Such description is performed by roto-translating the point cloud $\mathcal{P}$ so that its reference frame matches the one of SGURF. After the transformation, the points can be easily divided into 8 octants naturally defined by the signed axes:

$$(\vec{\mathbf{x}}_-, \vec{\mathbf{y}}_-, \vec{\mathbf{z}}_-) \cdots (\vec{\mathbf{x}}_+, \vec{\mathbf{y}}_-, \vec{\mathbf{z}}_-) \cdots (\vec{\mathbf{x}}_+, \vec{\mathbf{y}}_+, \vec{\mathbf{z}}_+) \tag{5.11}$$

Eight weights are computed and associated to each point in the cloud. These weights are calculated by placing three 1-dimensional Gaussian functions over each axis, centered at the cluster centroid and with $\sigma = 1cm$, which are combined by means of weight multiplications.

The weights associated with each point are binned into 8 histograms, each representing an octant. The index in the histograms is selected as $\frac{\bar{\mathbf{p}}}{R}$, where R is the maximum distance between any point and the cluster centroid.

Total size of these histograms is $8 \times 13 = 104$, thus the total size of the modified VFH is 303 bins. Figure 5.9(right) shows this histogram computed for a cluster found in a point cloud of a glass.

For compatibility and memory efficiency the modified histogram is still implemented in the Point Cloud Library [PCL] as a 308 components vector, thus the last five bins are left unused.

Finally, as in CVFH, the global OUR-CVFH feature is assembled (*step 4*) by concatenating all the previously found histograms together:

$$\mathbf{OUR-CVFH} = (\mathbf{VFH}^*_1, \mathbf{VFH}^*_2, \cdots \mathbf{VFH}^*_m) \tag{5.12}$$

where $\mathbf{VFH}^*_i$ denotes the modified version of VFH described above.

Computationally the OUR-CVFH feature estimation takes roughly as the CVFH, around $40ms$ for our data on our six-core machine, where the time spent mostly depends on the number of points in the cloud.

As a final note, OUR-CVFH and CVFH differs not only in the variants on clustering and VFHs estimation phases, but also in the cardinality of histograms created. For instance, even if both algorithms process the same

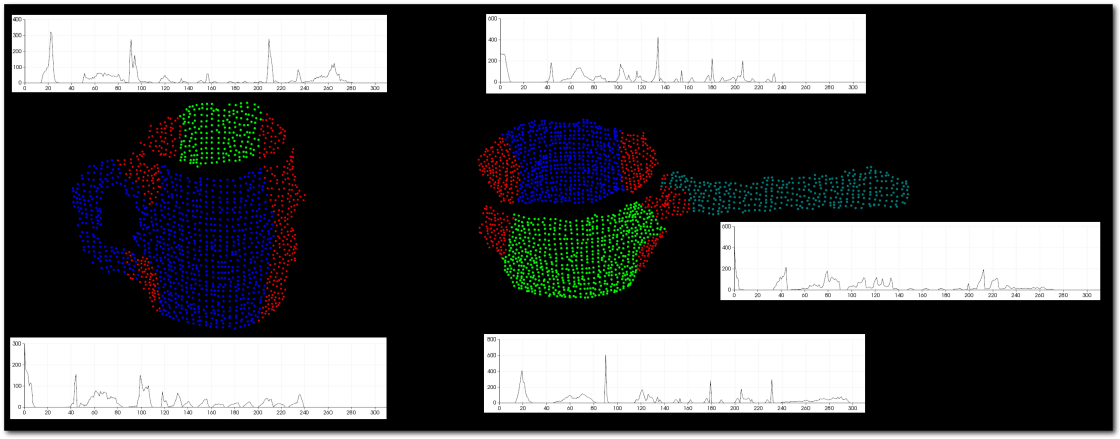


Figure 5.10: OUR-CVFH estimation of two objects: a mug (left) and a pan (right). Clusters are colored progressively from green to blue, while red points don't belong to any cluster. For every cluster the modified version of VFH is computed and shown next to them.

object and find the same clusters on it, OUR-CVFH may produce more histograms than the number of clusters found.

This may happen if the disambiguation phase of SGURF is unable to identify a single unique reference frame for a cluster, if this happens, the procedure is forced to produce a histogram for each reference frame found on the cluster, being unable to decide.

So the feature estimation may produce up to four histograms per cluster when the worst case of disambiguation happens. However this rarely happens, on our data less than the 1% of the feature estimated yielded more than one histogram per cluster.

Figure 5.10 shows the estimation of OUR-CVFH for two sample objects in the real database. The points not belonging to any cluster are colored in red, while the other colors represents the clusters found. Next to them the corresponding histogram is shown.

5.5 Matching and Comparison of Features

The matching and comparison of features is a widely used method to test feature efficiency and performance. In theory if one wants to compare two point clouds that represent objects to see if they are similar, he has to compare the corresponding features with some metrics.

The comparison is done in the feature space, thus it doesn't make sense to compare features of the same object, but originating from different methods, like for example VFH (Section 5.1) and ESF (Section 5.3). However a method for comparing features, to see if two objects are similar, needs to be used.

A common and natural method, also adopted here, consists of treating the histograms, representing the features, as points in n-dimensional space. Then estimate the distance between two points with some metric. If the two are "close", according to the chosen metric, then the two features are similar and thus also the objects; on the contrary it can be said that the two objects are different.

This leads to the problem of which metric to chose for the comparison, the feature's authors suggests the norms to use for each and we adopted these decisions in the thesis. They are:

VFH • Use the χ^2 distance:

$$\chi^2\left(\mathbf{VFH}_i, \mathbf{VFH}_j\right) = \sum_{k=1}^{308} \frac{\left(g_i^k - g_j^k\right)^2}{\left(g_i^k + g_j^k\right)} \tag{5.13}$$

ESF • use the L_2 distance:

$$L_2\left(\mathbf{ESF}_i, \mathbf{ESF}_j\right) = \sum_{k=1}^{640} \left(g_i^k - g_j^k\right)^2 \tag{5.14}$$

CVFH, OUR-CVFH • use a metric defined in [AVB+11] for both features:

$$\mathcal{D}\left(\mathbf{H}_i, \mathbf{H}_j\right) = 1 - \frac{1 + \sum_{k=1}^{308} min\left(g_i^k, g_j^k\right)}{1 + \sum_{k=1}^{308} max\left(g_i^k, g_j^k\right)} \tag{5.15}$$

where $\mathbf{H}$ denotes a histogram evaluated from a cluster of either CVFH or OUR-CVFH, and g_i^k indicates the k^{th} element of i^{th} histogram.

Once the metrics are defined, one should choose how "close" two features should be in order to declare if the objects they represent are similar or not. This threshold, not only is difficult to set, because different metrics are involved, but also is somewhat restrictive. In fact it classifies the problem into two fields: yes, they are similar; no they aren't. Losing the characterization of the features themselves.

Instead the following was done:

- Calculate the features of all the objects into a database (for example the synthetic database, see Section 3.1), obtaining four databases of features.
- Take a test point cloud from a database (the same or another), let's call it *query* or $\mathcal{Q}$ and estimate all the four features from it.
- Search the database of features for the k nearest ones (k-nearest neighbors) to $\mathcal{Q}$.
- These k candidates are sorted according to the smaller distance they have to $\mathcal{Q}$, obtaining a list of k candidates for each feature.

Suppose $\mathcal{L}_{VFH}$ is such list for the VFH feature, at first position (or rank 1) we find the candidate feature with the smallest distance to the query feature, among all features present in the database. Thus this object is the best approximation we could find in the database of known objects. Conversely at rank k the worst approximation is found. By having all the k candidates and their distances we have more informations on the nature of the *query* and we can build a "reasoning" to decide which one is the best match. However this will be discussed in detail in Chapter 6.

For semi-global features like CVFH and OUR-CVFH, the method for building a list of candidates is a bit more involved, because they can produce multiple histograms per object. To calculate the distance of semi-global features we elaborated the following:

- Produce the database of features, in which a single object can have multiple histograms, each associated to one of his clusters. A candidate object in this database, let us call it $\mathcal{O}$, has m clusters $\mathcal{O}_i$, where $i \in \{1, \cdots, m\}$.

- Estimate the features of the query, again multiple histograms are created, each associated to one of his clusters. Let us call the query object as $\mathcal{Q}$ and his n clusters as $\mathcal{Q}_j$, where $j \in \{1, \cdots, n\}$.
- For each $\mathcal{Q}_j$ search for the nearest neighbor among only clusters $\mathcal{O}_i$ of object $\mathcal{O}$. The nearest neighbors retrieved are n, then sum up their distances to gain a measure of similarity between $\mathcal{Q}$ and $\mathcal{O}$.
- Make a new entry in the list of candidates with the object $\mathcal{O}$ and its summed distance obtained before. Repeat the procedure until all objects in the database are processed.
- Sort the candidate list with minimum distance at top and truncate its size to k, so that it matches dimensions with the other global lists.

With this procedure the obtained list can be used as the other global ones. Figure 5.11 shows this procedure, with a graphical example.

To implement these procedures, a tool for fast indexing and searching was necessary. Fast Library for Approximate Nearest Neighbors (FLANN) [FLA] offers tools to build and index large datasets, for example KdTrees, and methods for extracting the nearest neighbors from such indexes. Experimentally FLANN can build a database of 1000+ features in a couple of seconds (building time is not important to our application, because is meant

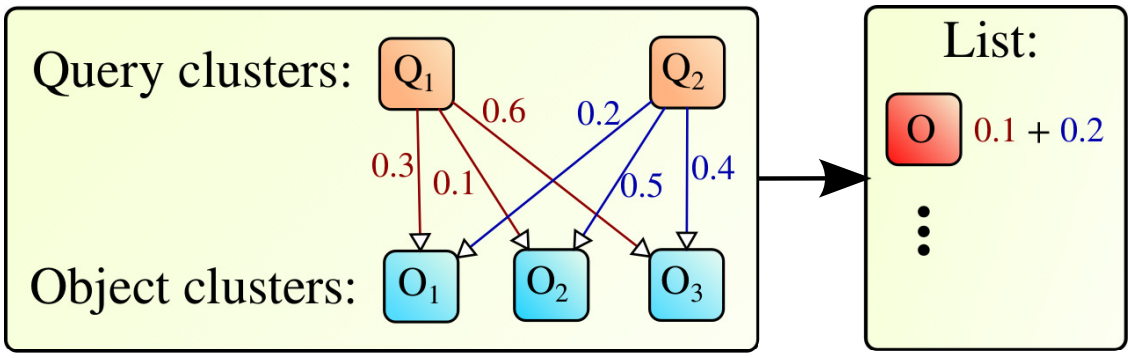


Figure 5.11: Illustration of semi-global features cluster information extraction, practical example. The query object has two clusters (orange), the candidate object has three (azure). Colored arrows represent the distance function between cluster features, each arrow label indicates the calculated distance. The list of candidates (right), gets a new entry for the object under examination (red) with a distance equal the sum of each minimum distance found. In this case $\mathcal{Q}_1 \rightarrow \mathcal{O}_2$, with distance of 0.1, and $\mathcal{Q}_2 \rightarrow \mathcal{O}_1$, with distance of 0.2. After all the objects in database have been processed, the candidate list gets resorted and resized.

to be done "offline") and retrieve up to 50 nearest neighbors in less than a millisecond.

To test the performances of each feature some experiments on the *synthetic database* were made. We focused on the ability of features to recognize known objects, to test the discriminative power of them. To this his end we built a database of features, starting from the synthetic scans, applied the preprocessing path described in Chapter 4 (MLS resampling, filtering, downsampling). The first outliers filter was removed for this test, because the synthetic scans are not affected by noise and thus they don't present outliers or imperfections along borders.

From the database, normals and the four features were estimated and indexed. Then each feature, from the corresponding database, was tested on the index, retrieving the 2 nearest matches. The purpose of the tests is to establish if the candidate in rank 2 is a feature obtained from the query object, not counting the pose, but just the object name. The candidate in rank 1 is obviously the query feature, because queries and database are taken from the same set of point clouds, so the features estimated are exactly the same, and therefore their distance in any metric is zero.

The results of these tests were organized in tables, one for each feature. They are visible in Tables 5.2, 5.3, 5.4 and 5.5. As one can see, the results of objects recognition are almost close to 100% for most objects.

Even if the data is synthetic, and thus prune of imperfections, it gives a good idea of the features' characterization. Even if some objects are similar, like for instance the mugs, the features are able to distinguish them from one another. Apparently the less accurate feature is VFH, while the other three fared almost equally. However it is not to be concluded that VFH is the feature with the poorest performances, because by changing the data, the performance can change drastically.

In the next chapter we focus on how we can use these features to achieve pose estimation for grasping, and how we can combine the performance of each feature to generally improve it.

Table 5.2: VFH objects recognition

Query	Matches		
	 (100%)		
	 (100%)		
	 (100%)		
	 (97.2%)	 (2.8%)	
	 (97.3%)	 (2.7%)	
	 (100%)		
	 (97.2%)	 (2.8%)	
	 (97.3%)	 (2.7%)	
	 (100%)		
	 (100%)		
	 (97.3%)	 (2.7%)	
	 (97.3%)	 (2.7%)	
	 (88.8%)	 (11.2%)	
	 (97.4%)	 (2.6%)	
	 (94.5%)	 (2.8%)	 (2.7%)
	(100%)		
	(100%)		
	(94.4%)	(5.6%)	
	(100%)		
	(100%)		

Table 5.3: ESF objects recognition

Query	Matches
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (98%)  (2%)
	 (100%)
	 (98.3%)  (1.7%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (98.2%)  (1.8%)
	(100%)
	(100%)

Table 5.4: CVFH objects recognition

Query	Matches	
	 (100%)	
	 (100%)	
	 (100%)	
	 (100%)	
	 (100%)	
	 (100%)	
	 (98%)	 (2%)
	 (100%)	
	 (100%)	
	 (100%)	
	 (98%)	 (2%)
	 (97.2%)	 (2.7%)
	 (97.2%)	 (2.7%)
	 (100%)	
	 (100%)	
	 (100%)	
	 (100%)	
	 (100%)	
	(100%)	
	(100%)	

Table 5.5: OUR-CVFH objects recognition

Query	Matches
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	 (91.6%)  (5.6%)  (2.8%)
	 (100%)
	 (94.4%)  (5.6%)
	 (100%)
	 (100%)
	 (100%)
	 (100%)
	(97.2%) (2.7%)
	(100%)

Pose Estimation

6

"A theory has only the alternative of being right or wrong. A model has a third possibility: it may be right, but irrelevant"

Manfred Eigen *(1927)*

Estimating the pose of different objects requires, not only the correct identification, but also the matching of the angle of acquisition. To this end we developed a database of object point clouds acquired with a rotating table (see Chapter 3), to simulate the various views a robot can face when trying to grasp the object.

To efficiently estimate the pose, a procedure has to correctly discern between different object and different poses of the same object. In Chapter 5 we showed the strong recognition value of global features to identify objects. However to achieve a robust pose estimation we need to test the features using real data and discern between different poses, not only objects.

In this chapter we discuss how we can use all the calculated features to achieve robust pose estimation and we show a procedure to efficiently estimate it with reasonable time.

6.1 Features Fusion

Global features are quite distinctive when discerning an object from another, but are as this effective when discerning poses? For example, can a feature distinguish a mug from the same exact mug rotated by ten degrees around an axis?

To answer this question numerous tests with the real database were performed. In summary, all the features have good performances in recognizing poses, not comparable to those in Section 5.5 however, but still more than acceptable. As also reported in authors papers [RBTH10, WV11, AVB+11, ATRV12], the performance of recognition greatly depends on the quality of training data.

It was our care to provide clean and neat acquisitions for the feature estimation, however this was not always possible due to the nature of some objects and sensor limitations.

It was our idea that the performances of recognition could be improved for our data and in effort to do that, a method for combining the answers (i.e. the list of candidates) of each feature was developed and explained here. The main base idea is to use all the features to recognize the same pose, so that if one gives the wrong answer, the other three might be right.

The first step was to build a *composite* list of candidates out of the four provided by the global features. Suppose a query pose needs to be identified on a database of objects in different poses. Each feature produces a list of k feasible candidates that best identify the query pose, ordered according to the distance from it. Let them be $\mathcal{L}_{VFH}$, $\mathcal{L}_{ESF}$, $\mathcal{L}_{CVFH}$ and $\mathcal{L}_{OURCVFH}$. Then the procedure to build the composite list is as follows:

1. Normalize all the distances ($\mathcal{D}^i$) in each list, so that the nearest candidate (at rank 1) has distance 0 and the farthest one (at rank k) has distance 1.

2. Take a candidate in a list and search for the same candidates in the others.

3. Make a new entry in the composite list for that candidate and set his distance to the average of distances in each list:

$$\bar{\mathcal{D}}^i = \frac{\mathcal{D}^i_{VFH} + \mathcal{D}^i_{ESF} + \mathcal{D}^i_{CVFH} + \mathcal{D}^i_{OURCVFH}}{4} \tag{6.1}$$

Then remove the candidate from $\mathcal{L}_{VFH}$, $\mathcal{L}_{ESF}$, $\mathcal{L}_{CVFH}$ and $\mathcal{L}_{OURCVFH}$.

4. Repeat *step 2-3* until all lists are empty.

5. Sort and resize the composite list so that it contains only the k nearest candidates to the query.

To achieve *step 1* the following was used:

$$\mathcal{D}_N^i = \frac{\mathcal{D}^i - \mathcal{D}^1}{\mathcal{D}^k - \mathcal{D}^1} \tag{6.2}$$

where $\mathcal{D}^i$ indicates the distance of the candidate at i^{th} rank, and $\mathcal{D}_N^i$ its normalized distance so that $\mathcal{D}_N^i \in [0,1]$.

This was necessary because different features uses different distance metric, see (5.13), (5.14) and (5.15), so by normalizing them in a range of $[0,1]$, we cloud make comparison between different metrics. As an additional note when computing $\bar{\mathcal{D}}^i$ (6.1), if a candidate is not present on a list, we treated his distance to be 1 (the worst possible). This penalizes sparse candidates that are not present on all the lists, because they most likely aren't the correct ones.

The approach of choosing candidates on which all features agree has proven to improve the general performance of matching for pose estimation. To support this claim, an example of features matching is represented in Table 6.1. A pose from the real database at 170° of a mug is taken as query. The four features produce lists of $k = 6$ candidates, from which the composite list is assembled. As one can see the latter put the correct pose at rank 1, while VFH putted it at rank 4 and ESF failed to identify it in the first 6 ranks.

In this example also the semi-global features correctly put the query at rank 1, but this is not always the case, in fact others example could be produced where those are wrong and others are right. The important thing here is that the composition will most likely have the correct answer among its ranks, if at least one feature finds it. To improve this chance, higher values of k can be used, but at the cost of more computational time. A good compromise for our data was to set $k = 20$, as shown further below the chapter.

To support this choice and show the performances of poses recognition, a few tests were performed:

Table 6.1: Example pose matching with real data

Rank	Query ⇒  170°							
	VFH		ESF		CVFH		OUR-CVFH	
	Cand.	Dist.	Cand.	Dist.	Cand.	Dist.	Cand.	Dist.
1	 200°	0	 10°	0	 170°	0	 170°	0
2	 140°	0.22	 110°	0.08	 10°	0.93	 150°	0.95
3	 50°	0.46	 200°	0.14	 160°	0.96	 160°	0.96
4	 170°	0.48	 350°	0.34	 160°	0.97	 190°	0.97
5	 210°	0.49	 340°	0.38	 350°	0.98	 350°	0.97
6	 20°	1	 50°	1	 0°	1	 10°	1

Rank	COMPOSITE	
	Cand.	Dist.
1	 170°	0.37
2	 10°	0.73
3	 200°	0.75
4	 110°	0.77
5	 200°	0.78
6	 140°	0.81

• Correct answers are highlighted in green.

•• Poses are represented by the object and the relative angle of the rotating table during acquisition. See Section 3.2 for more details.

Test 1. Synthetic data was used as database and all the poses from the real data were used as queries (a total of 2160 poses). Real data was also preprocessed to match the pipeline discussed in Chapter 4 (filter, resample, downsample). All the lists of candidates were computed, including the *composite* with $k = 40$. The results are presented in a graph picturing accumulated recognition rate versus rank. The percentages show the number of correct answers found in each rank, including the preceding ones. This means for example that a 70% recognition rate at rank 3 indicates that on 70% of the queries, the

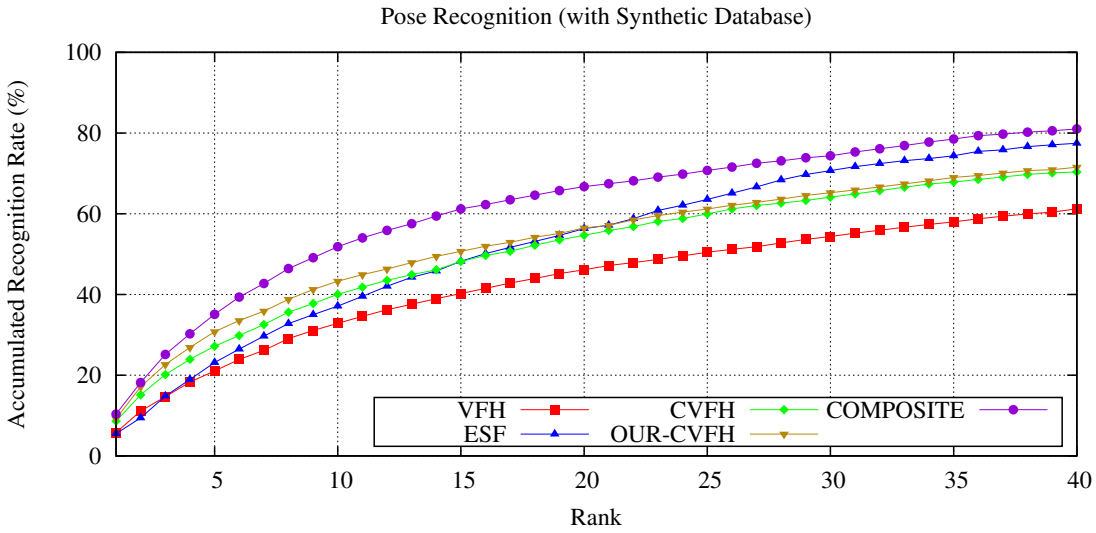


Figure 6.1: Accumulated recognition rate versus rank for the synthetic database and real queries. All the features and the composite list performance is shown.

correct answer was found either in rank 3, 2 or 1. The results are visible in Figure 6.1. The purpose of this test was to see if synthetic data can approximate reality with an acceptable degree of performance. However for $k = 20$ the recognition rate of the composite list is around 68%, which was not acceptable for our application. To reach acceptable recognitions, a greater k has to be used, but this can greatly increase the computational time of pose estimation. Although, if computational time is not a constraint, it is possible to use a synthetic database to efficiently estimates poses of real objects, saving incredible efforts to build the database, since synthetic data is easier and faster to produce.

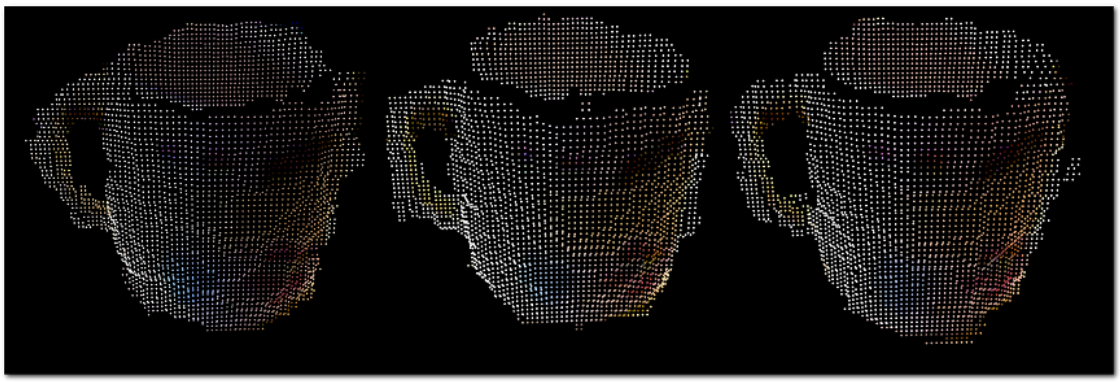


Figure 6.2: A pose of a mug from the three rounds of acquisition. Note the difference in the clouds, mostly around the borders and the handle. Point clouds are raw, meaning they are freshly acquired by the sensor and still have to be pre-processed.

Even so, one can see that the composite list improves the general recognition rate of each single feature.

Test 2. For this test, real data was used. Since three distinct rounds of acquisitions were made with the rotating table, real data is composed of 3 copies of each pose, taken at different moments in time. Even if the point clouds represent the same pose, they are distinct from one another, due to different lightening exposure, depth sensor noise and human error in positioning the object on the table. Figure 6.2 shows the same pose of a mug from the three rounds of acquisition. For this reason, different rounds of poses were used alternatively as database and queries, since each round is composed of 720 poses, the total number of tests performed was 4320. The nature of the test is similar to *test 1*, but the queries and database were preprocessed skipping the MLS resampling step. Results are shown in Figure 6.3, again as accumulated recognition rate versus rank. The composite list clearly outperforms the single features, with a recognition rate of 86.7% at rank 20.

Test 3. This test is exactly as *test 2*, but with a supplement of the MLS resampling during the preprocessing steps. We wanted to show the theoretical benefit of a surface resampling to features match, however only CVFH and OUR-CVFH seems to greatly benefict from this approach, while the performance of VFH is similar. ESF has slightly decreased his recognition rate at higher ranks (> 30). This leaves the composite list with an almost identical recognition rate as *test 2*. They are only slightly improved for smaller ranks, but CVFH outperforms it at higher ranks. Results are presented in Figure 6.4.

Ultimately the features and the composite list have shown good results in recognizing poses, with very fast computational time. Averagely with $k = 20$ the four list of candidates takes about $100ms$ to be assembled, while the composite list takes less than a millisecond. In the next section we discuss how we can choose the best candidate among the ones in the composite list to achieve a good estimation of the query pose.

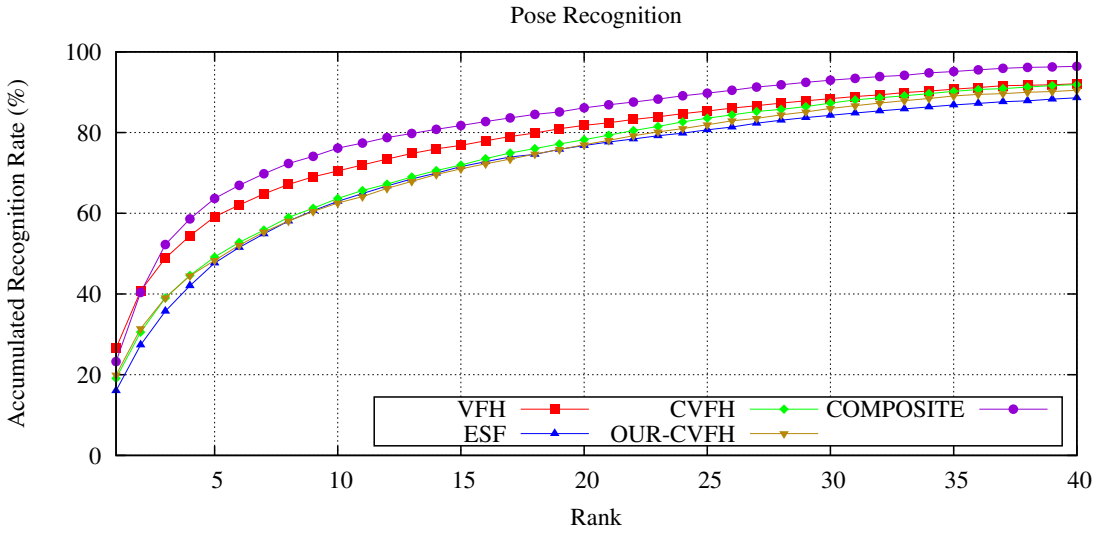


Figure 6.3: Accumulated Recognition Rate for real data, without MLS resampling. The composite list has the best recognition rate compared to the single features.

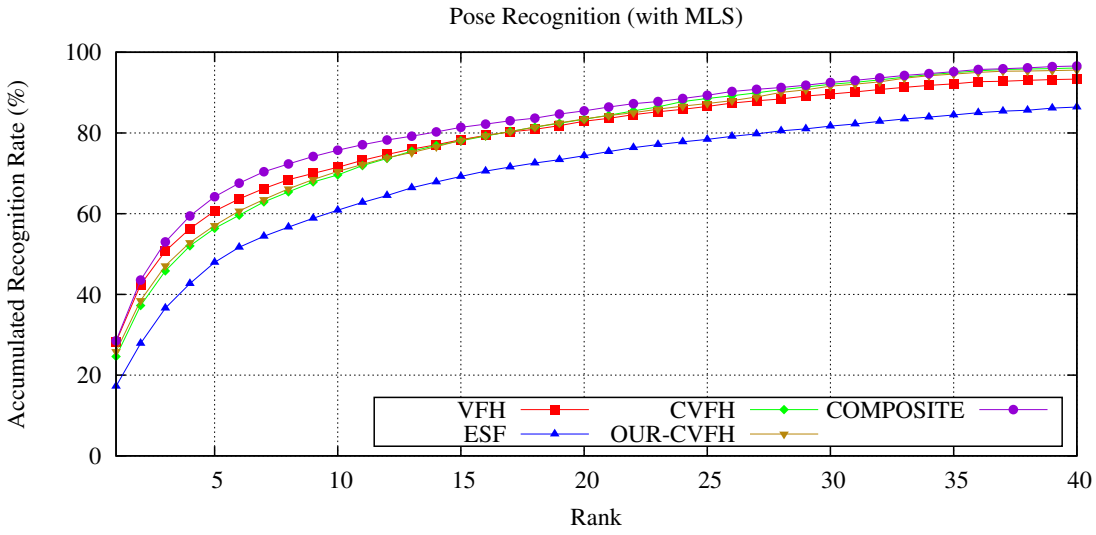


Figure 6.4: Accumulated Recognition Rate for real data with the use of MLS resampling.

6.2 Iterative Alignment

With the composite list assembled from the four global features, a method for choosing which candidate is the best approximation of the query pose needs to be assembled. The approach used in the thesis follows the common methods found in literature for pose estimation. That is to use an iterative algorithm to allign a candidate pose to the query, the candidate that minimize the error is most likely the correct estimation of the pose we seek.

The idea behind this is very simple, suppose a query pose needs to be estimated, then a good candidate would the one whose point cloud overlaps the query point cloud with minimal error. Then if a candidate can be aligned over the query so that they overlap almost perfectly, the transformation that brings the candidate over the query would be the estimation of the query pose in the candidate reference system. Of course this pose could be expressed in any reference system, as long as the transoformation between the candidates' system and the desired one is known, by applying a concatenation of transformations.

In this section we focus on how we can align a candidate over a query and decide which one is the closest approximation and therefore the correct pose estimation. Many algorithms and variants for alignment do exist, but the most famous and commonly used are: the Iterative Closest Point (ICP), first introduced in [BM92] and the Normal Distributions Transform (NDT), described in [BS03].

Iterative Closest Point align a point cloud, also called *source*, over another *target* cloud. For our application the *source* is a candidate cloud in the composite list, let's call it $\mathcal{P}_C$ and the *target* is the query cloud, be it $\mathcal{P}_Q$. The goal of ICP algorithm is to find the transformation, for which the error between the transformed points of $\mathcal{P}_C$ and the closest points of $\mathcal{P}_Q$ get minimal. This can be expressed with the following:

$$\underset{\mathbf{R},\mathbf{t},\mathbf{p}_j \in \mathcal{P}_Q}{min} \left(\sum_{\mathbf{p}_i \in \mathcal{P}_C} \|\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{p}_j\| \right) \tag{6.3}$$

where $\mathbf{R} \in SO(3)$ is a rotation matrix and $\mathbf{t}$ a translation vector.

ICP has three termination conditions:

- The root mean square error (RMSE) between points of $\mathcal{P}_C$ and $\mathcal{P}_Q$ has reached the user set threshold.
- The difference between transformations of two consecutive steps is smaller than a user set threshold.
- The algorithm has reached the maximum allowed number of steps.

By adjusting these conditions, the user can model the execution of the algorithm, rendering it more or less demanding. If one of the three conditions is verified ICP converges and the final transformation is presented with the relative RMSE. The algorithm however could converge at local minimum that leads to a false match.

Thus is extremely important to set "good" initial conditions, minimizing the chance of false convergences. The features matching and the composite list is a good approach to give ICP candidates that represents "good" initial conditions, because they are already similar to the query pose.

The other algorithm used is the Normal Distributions Transform (NDT), however, even if a bit more accurate in the output transformation, it is extremely slower than ICP. For our data, a full alignment of a candidate can take several minutes, rendering NDT not suitable for online pose estimation. We still used it in some tests to measure its efficiency for future offline works. NDT estimates a transformation that locally models the probability of measuring a point at a certain position by a collection of normal distributions. The transformation is then optimized with measures taken from the target. The full description of the algorithm is presented in [BS03]. Both algorithms have similar termination conditions, so they can be used interchangeably during the alignment procedure.

We left the pose estimation to the building of the composite list with its k candidates, that best approximate the query pose we need to find. Now to chose which one will be the best estimation we start aligning the candidates either with ICP or NDT, in the following way:

1. Set an RMSE threshold to be used as a termination condition, if a candidate has an RMSE below this threshold, it is to be considered a good estimation of the pose searched and the procedure terminates.
2. Align all the candidates in $\mathcal{L}_{COMPOSITE}$ starting from rank 1, but performs at most 5 steps of the alignment algorithm.

3. Reorder $\mathcal{L}_{COMPOSITE}$ according to minimum RMSE of candidates then resize it to half its size (rounding up), effectively discarding candidates with the worst RMSE.

4. Repeat *steps 2-3* until a candidate's RMSE falls below the threshold or only one candidate remains in $\mathcal{L}_{COMPOSITE}$.

5. The candidate remained, or the one below the threshold is the best approximation we could find for the query, thus the output transformation of ICP or NDT is the final pose estimation.

This iterative approach was preferred, over a brute force alignment, mostly to save computational time. In fact the alignment algorithm can be the bottleneck of the pose estimation application and its execution time closely depends on the number of points in the clouds. So the philosophy in this approach is: "the less alignment steps are needed, the more time is saved".

The brute force approach, previously used, was to start aligning the candidate at rank 1 and wait for it to converge under the RMSE threshold; if it doesn't, align the one at rank 2 and so on. Suppose however, the convergence happens at rank 10, all the candidates aligned before it, have consumed a lot of computational resources. With the iterative method, after 5 steps of alignment, the best candidates emerges, and thus is more productive to concentrate resources on them, instead of trying to align a candidate that would not have converged anyway.

6.3 Pose Estimation Performances

The pose estimation procedure described above, provides an estimation of the query object pose with a transformation that represents the *6 Degrees of Freedom* the object can have in three dimensional space. To prove its efficiency and correctness some tests were prepared and executed. The real database was used, with the acquisition rounds taking turns as database and the others as queries, thus totalling again 4320 pose estimations. For each query, the lists of candidates were built out of the feature estimation, the composite list assembled, and the list aligned with iterative process described above. Results are visible in Tables 6.2, 6.3 and 6.4. The tables report tests with different parameters used like the threshold for RMSE error and total length of the lists (k), all alignments were performed with ICP.

The final candidate chosen by the method, thus the estimation of the query pose, was classified as:

Success If the candidate and the query are exactly the same pose, thus same object and same angle on the rotating table.

Direct Neighbor If the candidate matches the query object, but the pose is not correct, however the angle is a direct neighbor of the query; i.e it is $\pm 10°$ from the correct one.

Same Object If the candidate is the same object as the query, but the pose is wrong and does not fall into the previous case.

False If the candidate and the query are different objects.

Each pose estimated and classified this way was also grouped according to the objects it tries to estimate. This gives a better understanding, because performances are quite varying depending on the type of objects.

This is most likely so, because of the quality of acquisitions. Generally speaking a big object with large surfaces (like the containers in the 2^{ND}, 3^{TH} and 4^{TH} row) is less affected by the sensor depth noise and resolution (which is roughly $3mm$), resulting in better shaped and defined surfaces. On the other hand, smal objects with narrow surfaces (like the spoons in the last three rows) suffers greatly from sensor noise and resolution, resulting in discontinued and not clean surfaces.

Table 6.2: Pose estimation results (RMSE threshold $3mm$ and $k = 20$)

Objects	Success (%)	Dir. Neigh. (%)	Same Obj. (%)	False (%)	Under Thresh. [a] (%)	Time [b] (ms)
	51.85	35.19	11.11	1.85	54.63	1026
	78.7	20.37	0.93	0	94.44	482
	92.59	7.41	0	0	95.83	1828
	75.93	22.69	1.39	0	93.98	790
	33.33	48.15	18.06	0.46	88.43	587
	34.26	50.46	12.96	2.31	82.41	548
	63.43	36.11	0.46	0	92.59	773
	46.3	44.91	8.8	0	95.83	887
	25.46	43.52	31.02	0	91.67	250
	73.15	25.46	1.39	0	96.3	727
	57.87	38.89	1.39	1.85	92.59	293
	56.94	38.89	3.24	0.93	95.83	228
	53.7	40.28	3.24	2.78	93.98	316
	50	43.98	2.78	3.24	99.07	205
	63.43	33.33	2.31	0.93	75.46	1345
	37.5	51.85	9.72	0.93	75	818
	35.19	51.39	12.5	0.93	62.04	1034
	18.06	25.93	47.22	8.8	9.72	1269
	22.33	35.81	33.95	7.91	20.47	1114
	41.2	39.81	17.13	1.86	35.65	1303
Average	**50.57**	**36.72**	**10.97**	**1.69**	**77.31**	**687.61**

a. Percentage of candidates chosen, because they fell under the RMSE error threshold.
b. Time is referred to the alignment phase only.
• Best object performance is highlighted in green, worst in orange. Symmetric objects around $\vec{y}$ in yellow.

Table 6.3: Pose estimation results (RMSE threshold 3.5mm and $k = 20$)

Objects	Success (%)	Dir. Neigh. (%)	Same Obj. (%)	False (%)	Under Thresh. [a] (%)	Time [b] (ms)
	50.46	36.57	11.57	1.39	73.61	689
	67.13	31.02	1.85	0	99.54	175
	80.56	18.98	0.46	0	99.54	829
	77.31	18.06	4.63	0	99.07	414
	35.19	46.3	18.52	0	92.59	335
	23.15	56.02	18.52	2.31	87.04	458
	54.63	44.44	0.93	0	96.3	454
	37.96	50.46	11.57	0	98.61	573
	21.3	37.96	40.74	0	93.98	212
	65.74	32.87	1.39	0	98.61	493
	46.76	45.37	4.17	3.7	98.15	163
	53.7	39.81	5.56	0.93	96.3	178
	42.59	49.07	5.56	2.78	99.07	166
	48.15	41.67	2.78	7.41	100	148
	56.48	40.74	1.85	0.93	87.04	787
	27.31	54.63	18.06	0	92.13	524
	28.7	53.24	17.13	0.93	81.48	737
	18.06	27.78	46.76	7.41	26.39	1121
	23.15	38.43	29.63	8.8	54.17	815
	49.54	33.33	15.28	1.86	61.11	900
Average	**45.39**	**39.84**	**12.85**	**1.88**	**86.74**	**442.22**

a. Percentage of candidates chosen, because they fell under the RMSE error threshold.
b. Time is referred to the alignment phase only.
• Best object performance is highlighted in green, worst in orange. Symmetric objects around $\vec{\mathbf{y}}$ in yellow.

Table 6.4: Pose estimation results (RMSE threshold $3mm$ and $k = 40$)

Objects	Success (%)	Dir. Neigh. (%)	Same Obj. (%)	False (%)	Under Thresh. [a] (%)	Time [b] (ms)
	54.63	31.48	11.57	2.31	27.78	2686
	77.31	22.22	0.46	0	86.11	1558
	93.52	6.48	0	0	82.41	6548
	70.83	26.85	2.31	0	80.56	2822
	29.17	51.39	19.44	0	81.02	1652
	55.56	41.2	3.24	0	86.11	1014
	65.28	34.26	0.46	0	87.04	1657
	49.07	42.13	8.8	0	83.33	2222
	34.26	48.15	17.13	0.46	98.15	301
	75.93	22.22	1.39	0.46	93.98	1356
	60.65	34.26	2.78	2.31	90.28	751
	63.43	31.02	3.24	2.31	92.13	566
	58.33	33.33	2.31	6.02	91.2	811
	58.33	36.57	0.93	4.17	95.83	483
	64.81	30.56	3.7	0.93	50	4278
	61.11	35.19	3.7	0	75.46	1976
	57.87	41.2	0.93	0	45.83	2904
	16.67	24.07	46.3	12.96	1.85	2640
	22.69	34.72	34.26	8.33	4.63	2558
	39.81	39.35	19.91	0.93	12.5	3035
Average	**55.46**	**33.33**	**9.14**	**2.06**	**68.31**	**1818.17**

a. Percentage of candidates chosen, because they fell under the RMSE error threshold.

b. Time is referred to the alignment phase only.

- Best object performance is highlighted in green, worst in orange. Symmetric objects around $\vec{\mathbf{y}}$ in yellow.

This discontinuity in real data, clearly affects the recognition capability of features, and as a consequence the pose estimation for those objects is not so accurate.

Some other objects are unaffected by rotation around $\vec{\mathbf{y}}$, because they are symmetric. For example if you look around a glass on a table from the same height, you will keep seeing the same cylindrical surfaces, no matter how you turn around. For these objects, highlighted in yellow, it doesn't make sense to distinguish between angles, so the estimation of the object pose is correct as long as the object itself is. Hence we considered a good result all the first three columns.

Another consideration about errors has to be made, by evaluating the results proposed, one has to consider the human error committed during the acquisition procedure. In fact even with the use of the rotating table, see Chapter 3, the object has to be put on the table by hand. Considering the use of a common reference frame, centered on the table, and that three distinct rounds of acquisitions were performed, an object pose of a round could be different from the same pose of a different round. For example the cup at 10° of the first round could be more similar to the cup at 20° of another round, rather than the one at 10°, due to errors in positioning the cup on the table during different rounds. These errors are intrinsically unavoidable and can lead to imperfect matches.

The matches are performed across different rounds of acquisition, because is more relevant to test recognition with similar point clouds, rather than identical ones. In reality during visual recognition phase of the grasping, it is impossible to have a point cloud identical to the one in the database of known poses, no matter how accurate and precise the sensor is.

For these reasons we included the *Direct Neighbor* result along with the others, because a pose of ±10° from the correct one might still be a good approximation of the seeked pose, given the nature of real data. Thus the sum of *Success* and *Direct Neighbor* percentages is considered a good result for us. With these considerations the average pose estimation is considered very accurate in 50% of the tests and a good approximation in 85% of them.

The last two columns of the tables presents, in order, the percentage of candidates that were chosen, because their RMSE error fell under the threshold, and the average time spent during the alignment phase. The first

is important to determine if the threshold set is too restrictive, but is also important to understand the quality of the estimations. A low percentage here means that the majority of candidates were chosen, because they were the only one left in the list at the end of iterative alignment. This could mean that the threshold set was too restrictive, or the candidates were too different from the query.

Either way is desirable to have that percentage has high as possible to reduce computational time: lots of candidates falling under the threshold will stop the iterative alignment sooner, saving time. The alignment time

Table 6.5: Pose estimation with various parameters

Align	Method	K	RMSE Thresh.	Pre Process [a]	Time [b] (*ms*)	Rate [c] (%)
ICP	Brute Force	20	0.003	F-U-D	2782.91	48.29, 37.69
ICP	Iterative	20	0.0035	F-U-D	1681.1	45.39, 39.84
ICP	Iterative	20	0.003	F-U-D	1926.21	50.57, 36.72
ICP	Iterative	40	0.003	F-U-D	3138.29	55.57, 33.33
ICP	Iterative	20	0.003	F-D	2058	55.63, 31.67
ICP	Iterative	10	0.003	F-U-D	1630	47.78, 35.49
ICP	Iterative	20	0.0025	F-U-D	2613.4	55.9, 32.92
NDT	Iterative	20	0.003	F-U-D	39700.2	42.57, 38.97
ICP	Iterative	20	0.0035	F-D	1477.25	48.96, 35.97

a. Where, *F* stands for outliers filtering, *U* for MLS resampling and *D* for voxelgrid downsampling.

c. Percentage of *Succes* followed by *Direct Neighbor* rate.

b. Time is referred to the total average time of whole procedure.

itself is also dependant from the total number of points in the cloud and the number of candidates (k) in the composite list.

In effort to summarize the various parameters used in the pose estimation procedure, Table 6.5 is proposed. It describes the variance of "good" recognition rate of poses and total execution time, over the various parameters and methods chosen. From the table results the following can be extrapolated:

- Increasing k slightly improves recognition rates, as expected, but at the expense of execution time. On the other hand, the inverse is obtained by lowering it, but there's a point, beyond which, lowering k will only lower the recognition rates, while gaining almost nothing in execution time.
- Increasing the RMSE threshold, the alignment is forced to converge earlier, lowering execution time, because fewer alignments have to be performed. However the convergence is most likely to happen for direct neighbors of the query, lowering the *Success* rate and raising the *Direct Neighbor*, though their sum is almost identical. Meaning that neither *Same Object*, neither *False* rates have increased much.
- Alignment with NDT yields almost the same performances as with ICP, but execution time is greatly increased, rendering it unsuitable for online applications.
- Iterative alignment improves both recognition rates and execution time, over brute force approach, thus it has to be preferred in all the situations.
- Resampling with MLS uniforms the surfaces, rendering them less distinctive from one pose to another, consequences are that *Success* rate is decreased in favor of *Direct Neighbor* rate, although their sum is almost identical. It increases preprocessing time, but alignment time is reduced, because smoother surfaces are better and quickly aligned, thus total execution time is almost the same.

Depending on performances required, the fastest approach would be to use no preprocessing (just downsampling), a small k like 10 and a threshold

greater than 0.0035. On the other hand to maximize recognition rate, at the expense of time, one has to increase k and reduce threshold.

A reasonable compromise would be with $k = 20$ and threshold of 0.003 yielding a recognition rate of 87% circa, while maintaining total execution time under $2s$.

The following sections present some adaptations and refinements of the Pose Estimation process, described so far, so that it could be used in various grasping environments. Particularly, within the Pacman project [Pac13], whose goals are to grasp known and unknown objects in clean and cluttered environments.

6.4 Pose Estimation in Arbitrary Reference Systems

The transformation matrix $\mathbf{T}_{4\times4}$, obtained from the final candidate during alignment, is a homogeneous transformation describing a roto-translation from the candidate to the query. Thus is a pose estimation of the query expressed in the candidate reference frame. By recalling Chapter 3, this reference frame was constructed over the rotating table and it is the same for all acquisitions. Now since the candidate and the query, during the tests, have an almost identical reference frame, different only in small human errors in positioning the object, $\mathbf{T}$ should be close to the Identity matrix $\mathbf{I}_{4\times4}$.

The pose expressed in this reference system is of little use, however suppose, that the user wants the pose expressed in another arbitrary reference frame, that could be the base of a robot, another table, or a completely arbitrary reference frame. Then all is needed is to know the transformation between the old frame centered in the object and the new wanted frame. $\mathbf{T}$ can then be concatenated with the inverse of such transformation obtaining the wanted pose in the chosen reference frame.

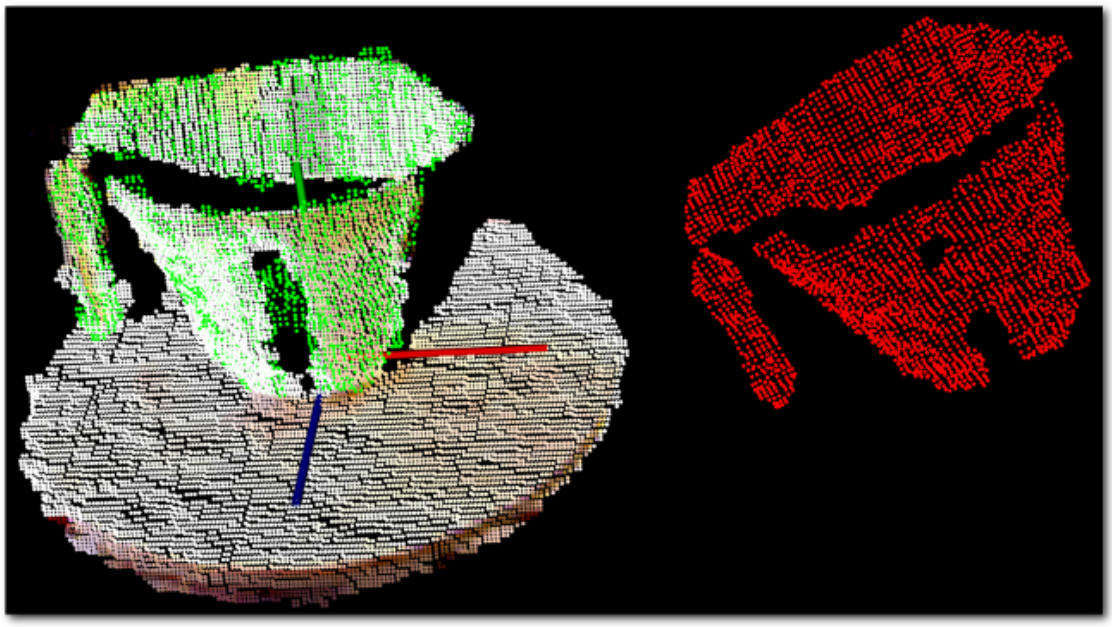


Figure 6.5: Pose estimation for a container. The object in the scene is the query, with his own color. In red the best candidate is chosen and displayed in the reference system we wanted: translated by $30cm$ along $\vec{\mathbf{x}}$ and rotated around $\vec{\mathbf{z}}$ by 30°. Candidate after the transformation is aligned on the scene and pictured in green.

Figure 6.5 shows an example of a container pose estimation. The trans-

formation is expressed in a reference system that was translated by $30cm$ along $\vec{\mathbf{x}}$ and rotated around $\vec{\mathbf{z}}$ by 30°. After the transformation the candidate is aligned on the scene and it is displayed in green.

In this case, the transformation that bring the unaligned candidate, in the chosen reference system, to the aligned one is the following:

$$\mathbf{T} = \begin{bmatrix} 0.861 & 0.512 & -0.001 & -0.286 \\ -0.512 & 0.86 & 0.031 & 0.152 \\ 0.016 & -0.026 & 0.999 & -0.002 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

where one can see the top-left submatrix $\mathbf{R}_{3\times3}$ is a rotation that is similar to an elementary rotation around $\vec{\mathbf{z}}$ and the last column vector $\mathbf{t}$ is the translation.

In definitive, with this technique, it is possible to generalize the pose obtained to any reference system in order to match any application requirements.

6.5 Pose Estimation with complete object models

The informations provided by the pose estimation lays not only in the transformation provided, but also in the point cloud itself. Once transformed the candidate cloud contains many informations that may be useful to an object grasping application. For instance the coordinates of all the points becomes known to the robot, because they are expressed in his reference system, since transformed. One could think of applying the transformation to a complete model of the object, so surfaces not directly visible in the current view, becomes approximated by the model. The robot could try to grasp those surfaces even if not directly visible, because the coordinates of all points on them are now known.

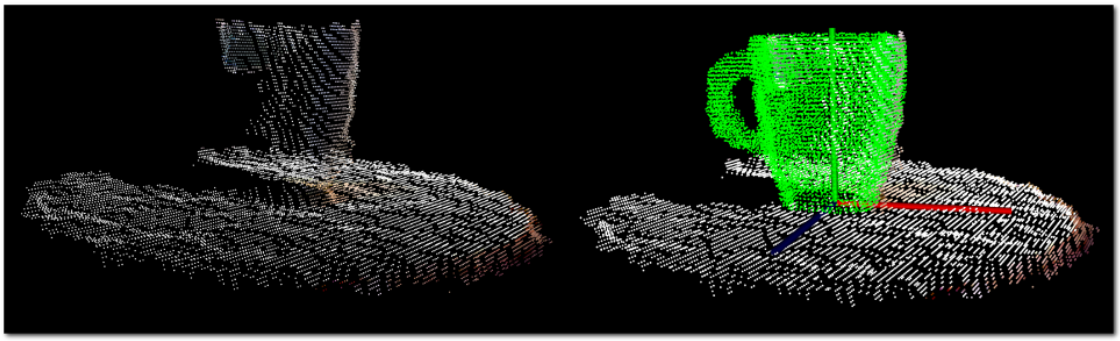


Figure 6.6: Use of an object model during pose estimation. The scene, in which the query pose lays (left), is oriented in a way that the sensor, located to the far right of the scene, is unable to see the mug handle. The query however gets correctly recognized as a mug and its pose transformation estimated (right). Object model is transformed with it and aligned on the scene. Now the robot knows that what it's seeing is a mug, and as such there's a handle behind it. The model provides a good approximation of the handle coordinates, so the robot could try to grasp it, even if it doesn't see it.

Figure 6.6 shows an example of such model usage with a mug. The complete object model was obtained in precedence from all the available poses of the mug, by registering them all in the same point cloud. The transformation found during pose estimation is applied to the mug model so that it aligns perfectly on the scene, the informations provided by the model can be later used for grasping.

In order to obtain a complete model of the object, all poses was refined with a few steps of ICP and incrementally added on top of each other. Then, since surfaces presents varying point density, due to overlapping, the model gets resampled with a voxel grid to uniform those surfaces. This process is

also known as *Registration*, Figure 6.7 shows the process in action.

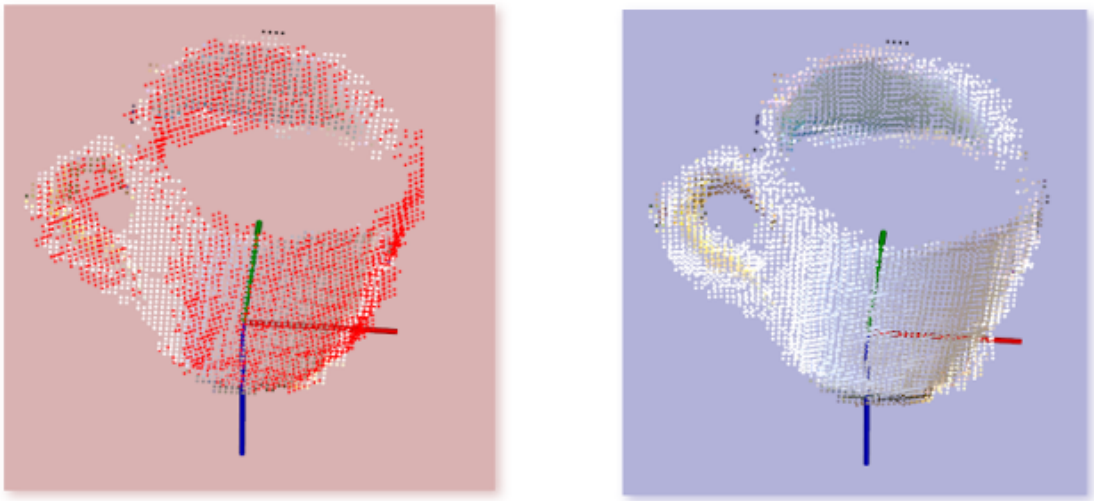


Figure 6.7: Registration of a mug poses. The point clouds gets aligned and incrementally added on top of each other (left), in red the current point cloud that is about to be added to the others. The so far composed model is shown on the right.

Another application could use the normals to the surfaces, suppose the robot knows how to grasp a specific object by placing its wrist near a point and normal to the surface. Then it could use the information provided by the normal estimations, already calculated and included in the point cloud, to align its wrist in the correct position.

6.6 Pose Estimation of Unknown or Cluttered Objects

A particularly interesting field in 3D recognition is pose estimation of objects that are not present in the current database. The ability to generalize with models that are not directly known to the robot, is a feature we are trying to pursue within the Pacman project [Pac13]. In order to understand what happens when the developed pose estimation procedure is asked to find the pose of an unknown object, we executed the following test:

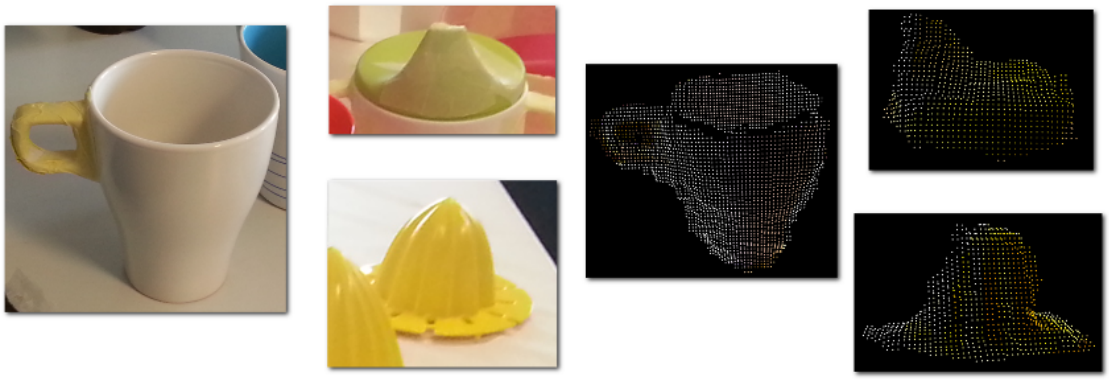


Figure 6.8: Three objects, not present in the database, used for testing: a mug (left), similar to the others, but with different handle, a juicer (bottom-left), smaller than the other and with a border at its base, and a green plug (top-left), that isn't really similar to anything inside the database. On the right three acquired poses of those objects are shown.

- The poses of three objects were acquired with the rotating table, following the same procedure of the other acquisitions. The objects are visible in Figure 6.8, there's a mug that is similar to the other mugs in the database, but with different profiles and different handle. A juicer, similar to the other one, but smaller and with a base border. A small green plug, that has nothing similar to, in the database. The total of 108 poses were acquired from them, divided in three distinct round of acquisitions, to match the database completeness.
- All the unknown clouds were submitted to the pose estimation process, and the final candidate results were recorded and organized in Table 6.6. Since we can't expect perfect alignment we relaxed the threshold parameter to 0.0045. The queries get classified with poses from objects in the database that best approximate them. The results were accumulated and grouped by objects.

Table 6.6: Pose estimation of unknown objects

Query	Accumulated Response *			
Plug	 93,5%	 3.7%	 1.8%	 0.9%
Juicer	 87.9%	 12.1%		
Mug	 53.7%	 28.7%	 17.7%	 1.85%

* How many times the query gets approximated with a pose from the listed objects.

Each row reports how many poses, in percentage, from the pictured objects were chosen as an approximation of the queries. The mug, as expected, was approximated mostly with other mugs, while in some cases with a glass. This is most likely due to the handle being invisible in some poses, without the handle, the mug is probably more similar to a glass in the features space.

The juicer gets approximated mostly with the other juicer or the funnel, however since both those objects are considerably bigger than it, the alignment is not precise. The green plug object is the most difficult to estimate, because is not similar to anything in the database. However most of the time it gets approximated with the juicer or the other plug. Even here, since those objects can not really fit, only a part of them is aligned on the query and thus the overall pose estimation is not so accurate. Some example pose estimation for the unknown objects are visible in Figure 6.9.

Another pursued feature of our pose estimation, is the ability to recognized poses in a cluttered environment, where objects are occluded and only parts of them are visible. This is particularly important, because the robot should handle most of the situations a human is facing, and thus it's not expected to always work in clean environments.

To test the pose estimation robustness to cluttered environments, we acquired several scenes with occluded objects and made some experiments. The query objects, extracted from the scenes, are the same as those in the database, but parts of them are missing. For these estimations we used a threshold of 0.005 and $k = 40$ to favor recognition at the expense of execution time. Some objects are still recognized, and their pose is correct, while some others have incorrect poses.

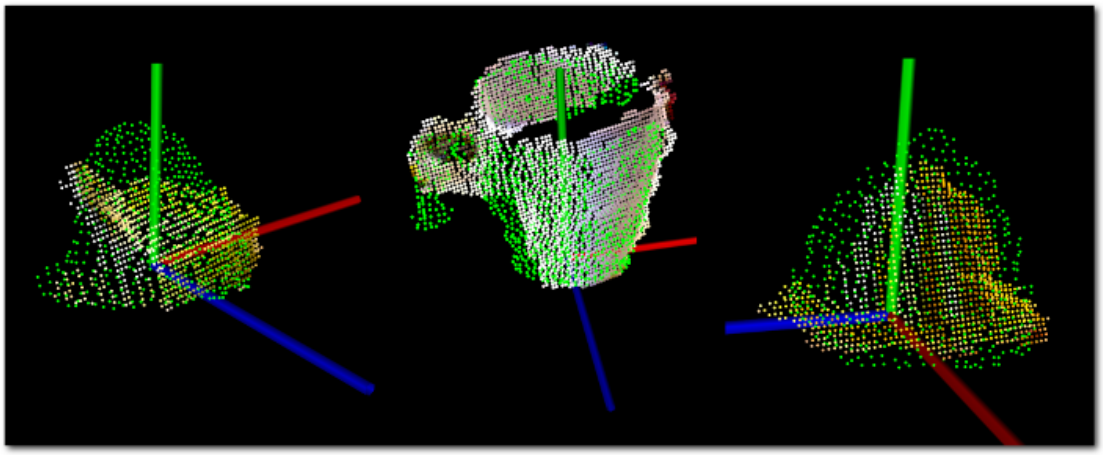


Figure 6.9: Example pose estimations for the unknown objects. Chosen and aligned candidates are pictured in green. The plug (left) gets approximated with the juicer, however since it is bigger only the borders are really close to the query and the overall alignment looks wrong. The mug (centre) gets approximated with another mug, the overall alignment is quite correct, only the handle exceeds from the query. Small juicer (right), gets approximated with its bigger counterpart, again only the borders fit, but the overall alignment is quite satisfactory.

Some results of the experiment are visible in Figures 6.10 and 6.11. From the cluttered scenes a query is extracted and aligned, the kettle, pan and juicer are estimated correctly, while the other three are not. The kettle, juicer and pan are only partially covered and their distinct shapes are still visible, probably because of this they are correctly recognized and aligned,

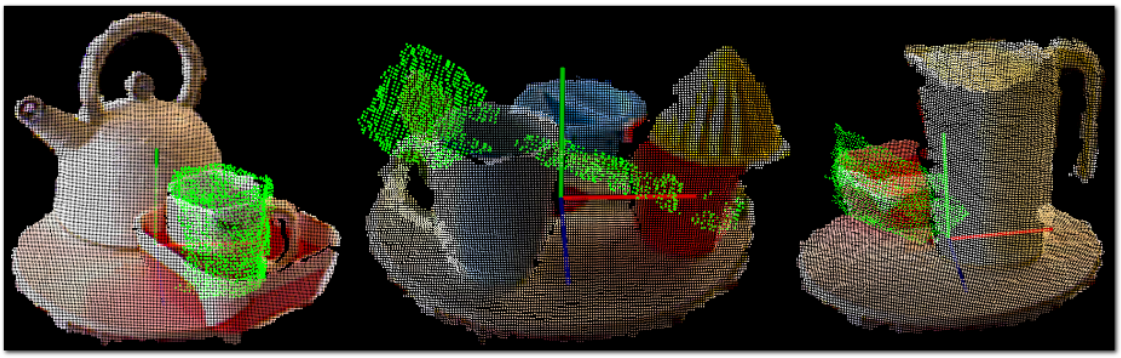


Figure 6.10: Wrong pose estimations in cluttered environments. Chosen and aligned candidates are colored in green. The first query, a mug (left), is inside a container, although it is correctly identified as a mug, its pose is wrong because is missing the handle. The spoon (middle) is almost completely occluded by other objects, its alignment is almost correct, however it is identified as a different spoon, thus the pose is wrong. Container (right) is occluded by the jug, it is correctly identified as a container, but the pose is wrong and not aligned.

the pan in particular has its handle completely exposed and that is the characteristic that distinguish it from other cylindrical containers in the database.

For the other examples proposed, the procedure has failed to correctly estimate the pose. For the spoon and container, probably because they are too cluttered by other objects and, the first, is recognized as a different spoon, thus the resulting alignment is wrong, the second, is correctly identified as a container, but the chosen pose is wrong and doesn't fit in the scene. Instead for the mug, the object is correctly identified, but the pose chosen doesn't have a handle, because is from a view from which it results behind the cup, thus the overall pose is wrong.

In definitive, working in cluttered environments or with unknown objects pose a serious problem to recognition and pose estimation. The procedure presented can safely handle lightly cluttered objects if their distinctive characteristics are visible and can address unknown objects if they are mostly similar to others in the database. We reserve further studies to improve the overall robustness of the procedure in presence of these scenarios.

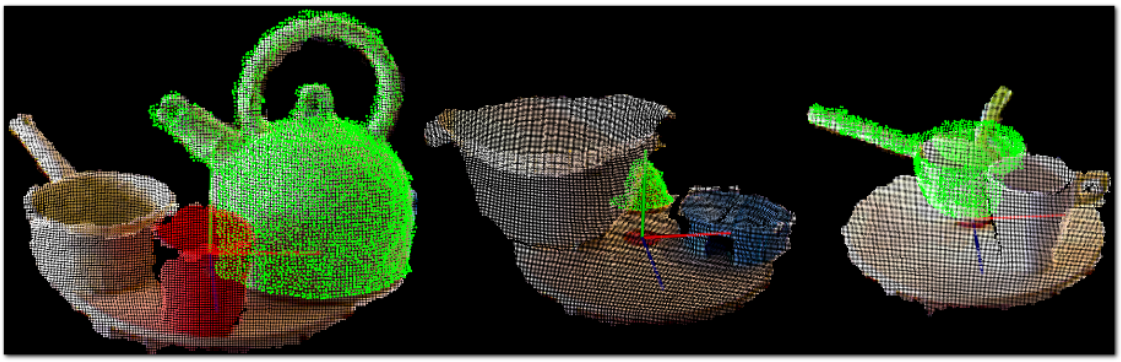


Figure 6.11: Correct pose estimations in cluttered environments. Chosen and aligned candidates are colored in green. The kettle (left) is partially occluded by a plastic glass, but it is correctly identified and its pose estimated. The juicer (middle) is partially behind a container, however its pose estimation is correct. Pan (right) is covered by a mug and has a spoon inside it, but from its visible parts the object is identified and the pose is correctly aligned.

7 Conclusion

"A conclusion is the place where you get tired of thinking"

Arthur Bloch *(1948)*

As robotics takes steps forward towards the creation of autonomous systems that can integrate with humans, the need to find ways of functioning in living environments, navigating and interacting with the surroundings, becomes a necessity. This means a robot should be able of mapping the environment in which it operates, by means of haptic capabilities and 3D perception. The latter, in particular, is a field of constant evolution where new technologies and complex routines are constantly being developed.

A robot equipped with such capabilities could operate in the environment, by recognizing objects, grasping and manipulating them. Therefore one of the most important aspect of autonomous robots is represented by the ability to perceive and understand objects present in the real world. The introduced Pacman project aims at building a robot that can accomplish such tasks, in particular object grasping and manipulation. So, along with strong haptic capabilities, the robot should be able to recognize and localize objects in his environment.

This thesis, being part of the project, proposed a procedure than can be used to accomplish such task, known as pose estimation. The first part of the thesis focused on data acquisition and manipulation, in particular in Chapter 2 we covered a semantic of 3D representations, needed to understand how 3D data is presented. In Chapter 3 we described how we acquired data to gain a repository of objects models used to test our pose estimation procedure. The database was necessary to give the robot a limited, but later improvable, understanding of the most common objects it

could grasp. The objects were acquired from different viewpoints, in order to simulate the possible views the robot could face in a real environment.

In Chapter 4 we discussed how we can manipulate that data, in order to improve its quality and partially address the imperfections and limitations of the sensor. Several processed data was generated to try to obtain better performances for our pose estimation procedure.

The second part of the thesis focused on how we can extract features from models of object poses, in order to obtain a mapping that can be matched to find correspondences between poses. We show the strong recognition capabilities of the features we chose to use, Chapter 5, and efficiently combined them to obtain a more robust evaluation in Chapter 6.

The procedure we proposed uses this combined evaluation to align the best candidates it could find to the unknown pose we wanted to estimate, in order to obtain a final candidate that best approximates the sought pose. We showed the good performances of this procedure in different scenarios, like a normal and clean environment, with objects not present in the database or in presence of occlusion. We sought to improve the robustness of the procedure with various parameters and pre-processing steps.

A closer look on execution time was always given throughout the thesis, because we wanted the procedure to be as fast as possible, so it could be executed in real time, if needed. We presented several results and tests to measure performances and we saw how several factors can improve or degrade the pose estimation:

The threshold set during the alignment provides a measure of how accurate we wanted the final pose estimation. It was shown it needed to be accurately set based on the quality of data and the type of application.

The total number of candidates in each list also affects performances. Generally speaking, the more candidates there are, the better, but a compromise needed to be found, in order to not compromise execution time too much.

The pre-processing pipeline chosen can influence the overall quality of data and consequently the pose estimation performance. We saw the importance of having the same point density through all the data, so that features could be consistent, also reducing the overall quantity of

points, by means of downsampling, can be beneficial to execution time. We discussed the beneficial effect of an outliers filter to acquired data, while the resampling with Moving Least Squares was not so performance improving as we first expected. Overall the best performance and execution time was obtained with just a filter and downsample pre-processing, because MLS resampling renders surfaces less distinctive between each other, lowering recognition of poses, but improving recognition of objects. However different kind of data may benefit more from different pipelines.

The algorithm during alignment needed to be carefully chosen to accomplish good performance and execution time. It was found that Iterative Closest Point offers the fastest alignment for an online application, while Normal Distributions Transform could be used in off line applications to gain a small benefit in alignment precision.

The quality of data can drastically change the overall performance of the procedure. While we tried to mitigate the sensor noise and imperfect measures, by means of covering sensible parts with opaque films, small and narrow objects still retains imprecise point clouds.

The last factor will most likely be addressed with time, as new and better sensors will be developed. The others, however are intrinsic to the procedure proposed and need to addressed with careful planning depending on the kind of application needed.

Despite the encouraging results presented in the thesis, there are still some issues that remains open for future research:

- We talked about quality of data that needed to be improved to obtain better results with the pose estimation. The data is affected by depth noise, most likely dependent by lightening conditions and the natural reflectiveness of some surfaces. We also saw how a small or narrow surface could result in a bad quality acquisition, perhaps a way to partially overcome this issue a dull opaque painting over those problematic surfaces.

- During the development of the thesis we acquired numerous point clouds of objects from 36 viewpoints around it, however a way to

improve this number is necessary for the advancement of the project. We though of mounting the sensor on the robot arm, so that one can control the acquisition process by setting the height and distance from the centre of the rotating table. Effectively increasing the number of poses per object and expand the database to cover more realistic situations.

- The number of objects present in the database should also be increased to give more generalization for a wider range of objects. This would increase the chance of positive recognition with an unfamiliar object, that is similar to others in database.
- Perhaps the procedure of pose estimation can be improved with the use of more features, even *local* ones, and a more sophisticate combination procedure. A parallel architecture could be designed, where all the features are calculated simultaneously by different threads, this would certainly at least improve the computational time of the whole procedure.
- All the algorithms used in the thesis could be redesigned to work with *Graphical Processing Units* (GPU) to obtain more computational power and most likely improve the overall execution time.

Though these are important aspects that could be addressed in future research initiatives, we are confident that in this thesis we managed to take small steps towards the realization of the Pacman project and towards the realization of autonomous grasping robots.

Bibliography

[ABCO+03] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 9(1):3–15, Jan 2003.

[ATRV12] Aitor Aldoma, Federico Tombari, RaduBogdan Rusu, and Markus Vincze. Our-cvfh – oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6dof pose estimation. In Axel Pinz, Thomas Pock, Horst Bischof, and Franz Leberl, editors, *Pattern Recognition*, volume 7476 of *Lecture Notes in Computer Science*, pages 113–122. Springer Berlin Heidelberg, 2012.

[AVB+11] A Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R.B. Rusu, and G. Bradski. Cad-model recognition and 6dof pose estimation using 3d cues. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 585–592, Nov 2011.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992.

[BNRV05] Piotr Breitkopf, Hakim Naceur, Alain Rassineux, and Pierre Villon. Moving least squares response surface approximation: Formulation and metal forming applications. *Computers & Structures*, 83(17–18):1411 – 1428, 2005. Advances in Meshfree Methods.

[BS03] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Intelligent Robots*

and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, volume 3, pages 2743–2748 vol.3, Oct 2003.

[Eas13] Easel. Expressive agents for symbiotic education and learning. http://easel.upf.edu/, 2013.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FLA] FLANN. Fast library for approximate nearest neighbors. http://www.cs.ubc.ca/research/flann/.

[Int14] Interaction. Interaction project. http://cms.interaction4stroke.eu/drupal/, 2014.

[Joh97] Andrew Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[Ope] OpenMP. Open microprocessing. http://openmp.org/wp/.

[Pac13] Pacman. Probabilistic and compositional representations for object manipulation. http://www.pacman-project.eu/, 2013.

[PCL] PCL. Point cloud library (pcl). http://pointclouds.org/.

[RBTH10] R.B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162, Oct 2010.

[RMBB08a] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent Point Feature Histograms for 3D Point Clouds. In *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10), Baden-Baden, Germany*, 2008.

[RMBB08b] R.B. Rusu, Z.C. Marton, N. Blodow, and M. Beetz. Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 643–650, Dec 2008.

[Rus10] RaduBogdan Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI - Künstliche Intelligenz*, 24(4):345–348, 2010.

[TSDS10] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, pages 356–369, Berlin, Heidelberg, 2010. Springer-Verlag.

[VTK] VTK. The visualization toolkit (vtk). http://www.vtk.org/.

[WV11] W. Wohlkinger and M. Vincze. Ensemble of shape functions for 3d object classification. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2987–2992, Dec 2011.